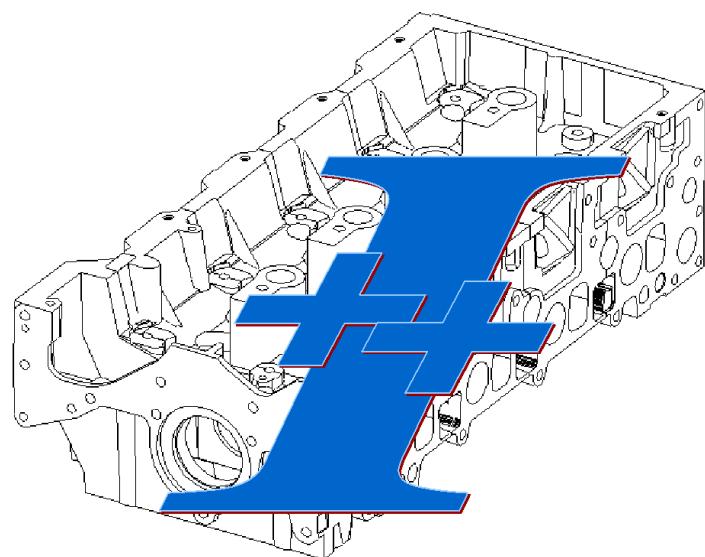




DAIMLERCHRYSLER

VOLVO



DME – Interface

Release 0.9

Contents:

1 I++ WORKING GROUP INFORMATION	5
1.1 This specification was created with the assistance of.....	5
1.2 The goal	5
1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment)..	5
1.4 Requirement.....	5
1.5 What is the intention of the specification ?	5
1.6 Schedule steps.....	6
1.7 History	6
2 PHYSICAL SYSTEM LAYOUT	7
2.1 DME-Interface Implementations	8
2.2 DME-Interface Model.....	8
2.3 Logical System Layout.....	9
2.4 DME-Interface and Subsystems	10
2.4.1 Monitor	10
2.4.2 Diagnostics.....	10
2.4.3 Application.....	10
2.4.4 Info.....	10
3 HIERARCHY OF COMMUNICATION	11
3.1 Layers	11
3.2 Examples of basic use cases	12
3.2.1 Sequence Diagram: Connect, Disconnect (Session).....	12
3.2.2 Sequence Diagram: Synchronous Communication	12
3.2.3 Sequence Diagram: Asynchronous Communication (Single Shot Events)	13
3.2.4 Sequence Diagram: Asynchronous Communication (Multiple Shot Events).....	13
3.2.5 Sequence Diagram: Handling of Unsolicited Errors	14
4 EVENTS	15
4.1 Transaction events, syntax.....	15
4.2 One shot events.....	15
4.3 Multiple shot events	15

4.4	Server events	15
5	OBJECT MODEL	16
5.1	Explanation	16
5.2	Reduced Object Model.....	17
5.3	Full Object Model	18
5.4	Packaging for visualization, see header files in subdirectories chapter 9	19
5.5	Contents of server	19
5.6	Contents of dme	20
5.7	Contents of cartcmm.....	20
5.8	Contents of cartcmmwithrotarytable.....	21
5.9	Contents of toolchanger	22
5.10	Contents of lib and unspecified.....	23
6	PROTOCOL	24
6.1	Communication	24
6.1.1	Character set.....	24
6.1.2	Initialization	24
6.1.3	Shutdown	24
6.2	Protocol Basics	25
6.2.1	Tags.....	25
6.2.2	General line layout.....	26
6.2.3	Transactions	27
6.2.4	Events.....	28
6.2.5	Errors	29
6.3	Method Syntax	30
6.3.1	Server Methods.....	30
6.3.2	DME Methods	33
6.3.3	CartCMM Methods.....	36
7	ADDITIONAL DIALOG EXAMPLES	44
7.1	Connect.....	44
7.2	Move 1 axis	44
7.3	Probe 1 axis	44
7.4	Move more axis in workpiece coordinate system.....	45

7.5	Probe with more axis	45
7.6	Set property	45
7.7	Get, read property.....	46
8	ERROR HANDLING.....	47
8.1	Classification of Errors	47
8.2	F1: Error severity classification	47
8.3	List of I++ predefined errors	47
9	C++ AND HEADER FILES FOR EXPLANATION.....	48
9.1	\main\main.cpp.....	48
9.2	\server	48
9.2.1	\server\server.h	48
9.2.2	\server\part.h.....	49
9.2.3	\server\server.cpp.....	49
9.3	\dme	51
9.3.1	\dem\dme.h.....	51
9.4	\cartcmm.....	52
9.4.1	\cartcmm\cartcmm.h	52
9.4.2	\cartcmm\eulerw.cpp	53
9.4.3	\cartcmmwithrottbl\cartcmmwithrottbl.h.....	54
9.5	\toolchanger	54
9.5.1	\toolchanger\toolchanger.h.....	54
9.5.2	\toolchanger\tool.h.....	56
9.5.3	\toolchanger\toolab.h	57
9.5.4	\toolchanger\toolabc.h.....	57
9.5.5	\toolchanger\gotoparams.h	58
9.5.6	\toolchanger\ptmeaspars.h.....	58
9.5.7	\toolchanger\param.h	59
9.6	Most important of lib	60
9.6.1	\lib\axis.h.....	60
9.6.2	\lib\eulerw.h	60
9.6.3	\lib>tag.h	61
9.6.4	\lib\ipptypedef.h	61
9.6.5	\lib\ippbaseclasses.h	62

1 I++ Working Group Information

1.1 This specification was created with the assistance of

Hans-Martin Biedenbach,	AUDI AG
Josef Brunner,	BMW
Kai Gläsner,	DaimlerChrysler
Dr. Günter Moritz,	Messtechnik Wetzlar
Jörg Pfeifle,	DaimlerChrysler
Josef Resch,	Zeiss IMT

I++ is a working group of five European Car manufacturers (Audi, BMW, DaimlerChrysler, VW and Volvo).

1.2 The goal

The I++ working group defined a requirement specification with the goal to achieve a new programming system for inspection devices. (not only for CMM's)

This specification will describe the I++ application protocol for the following types of DME's:

- Ø 3D coordinate measuring machines including multiple carriage mode
- Ø Form testers
- Ø Camshaft, crankshaft measuring machines

The spec allows to access all basic DME functionality.

1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment)

I++ turn one's attention to the difficulties of the interfaces. So I++ defined a team, who are responsible to workout a requirement specification for a neutral I++ DME interface.

1.4 Requirement

We demand a clear definition, that the DME vendor is responsible for the accuracy of his measurement equipment, in the sense that all necessary functions related to the equipment accuracy have to be implemented in the neutral I++ DME interface.

1.5 What is the intention of the specification ?

The following requirement specification is capable of further developments, this means the specification is valid for CMM's as well as other measurement equipments.

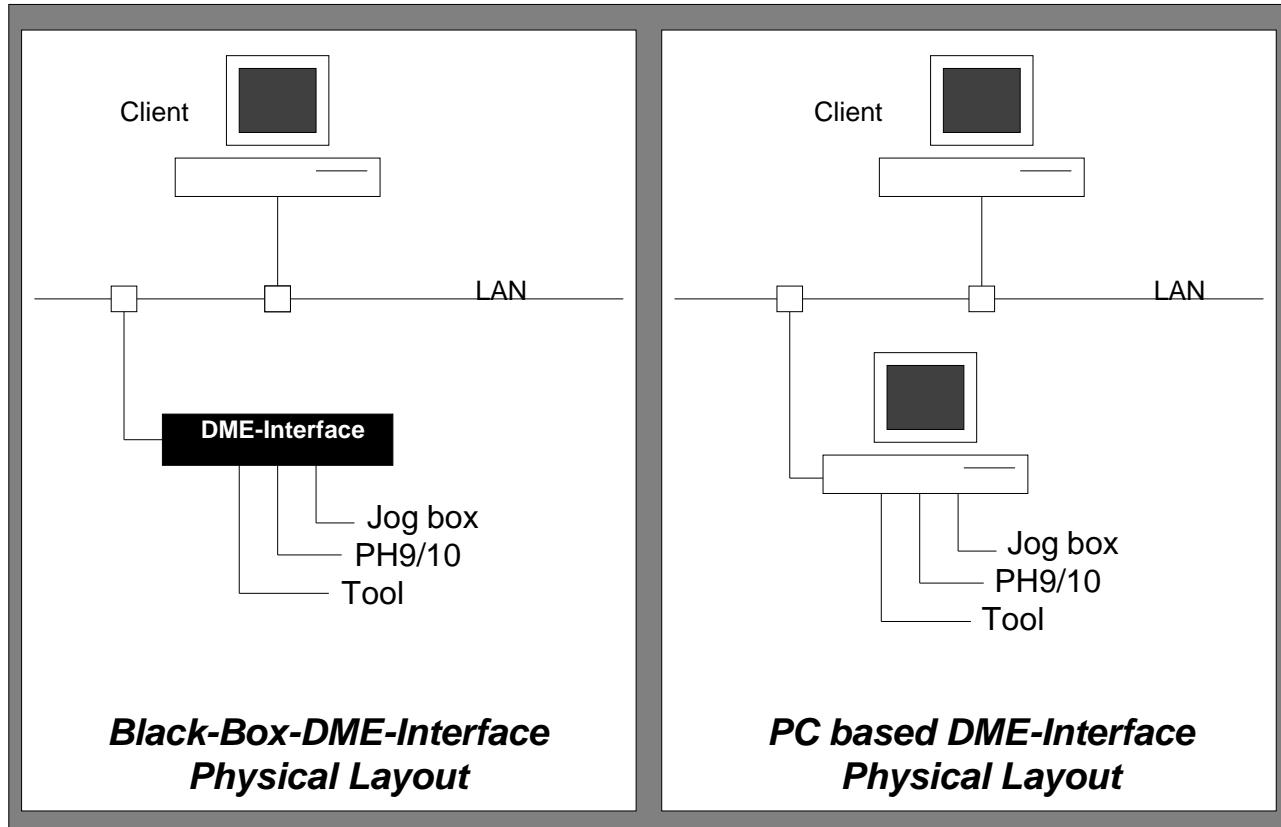
1.6 Schedule steps

Changes from 1.0 to 1.1 :	Multiple arms
Changes from 1.1 to 1.2 :	Scanning
Changes from 1.2 to 1.3 :	Rotary table
Changes from 1.3 to 1.4 :	Form testers
Changes from 1.4 to 1.5 :	Camshaft, crankshaft measuring machines

1.7 History

2 Physical System Layout

This section is intended for helping to explain the context of this specification. It is not part of the specification.



Picture 1 shows two examples of the physical system layout for these types of machines. In both examples the main components are Client computer (Client) and DME-Interface. Machine (including frame, motors, scales, ...)

Client and DME-Interface are connected through a local area network (LAN). Both client and DME-Interface use TCP/IP sockets for communication. The client computer runs the application software for the measurement task. The DME-Interface implements all functionality required to drive the machine. The application software on the client talks to the DME-Interface in order to execute elementary measurement tasks (picking points, scanning, ...). This specification describes the protocol that the client uses to run the machine through the DME-Interface.

2.1 DME-Interface Implementations

The main difference between the two implementations of the DME-Interface in Picture 1 is the physical implementation of the DME-Interface, which is

- PC based or
- “Black Box” based

While PC based DME-Interface provide a direct physical (screen, keyboard) user interface the black box based system provides no direct user interface.

PC based systems may provide additional low-level user interfaces that help the user to control and monitor the machine.

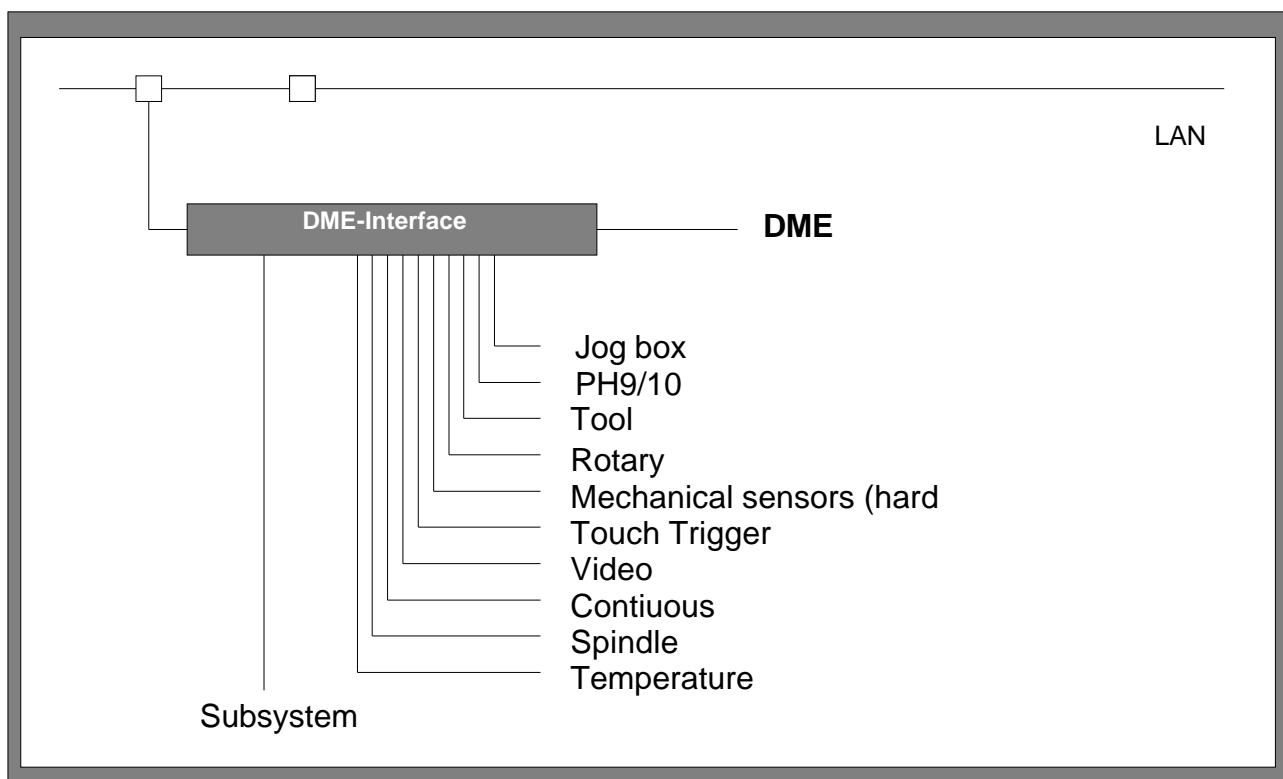
“Black Box” based system have a potential cost advantage.

2.2 DME-Interface Model

Picture 2 shows the system layout we will use in this document for explanations.

It is important to recognize, that all subsystems are linked to the DME-Interface.

This implies, that the client must use the protocol, to access subsystem functionalities, like rotating a PH10.



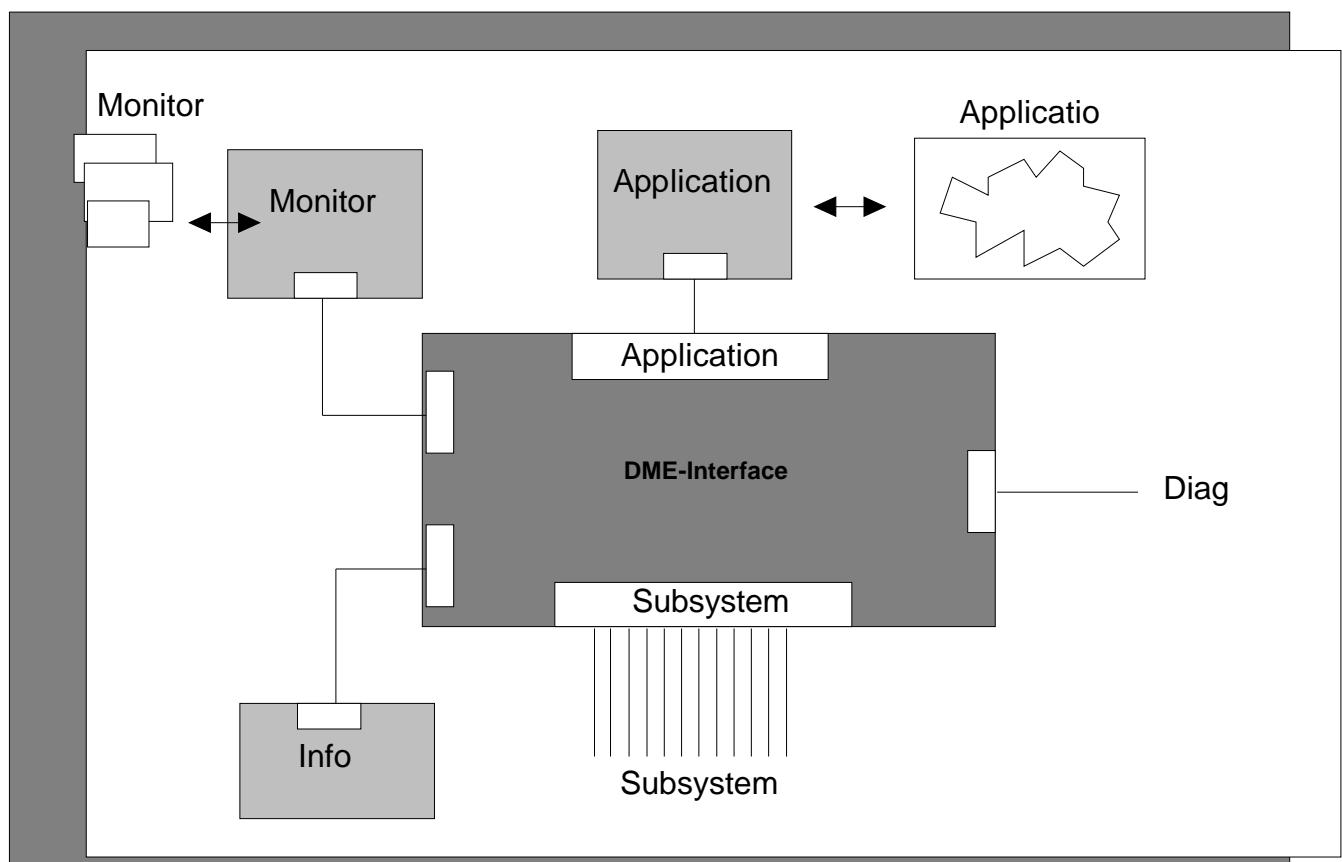
2.3 Logical System Layout

Picture 3 shows the logical layout of the system with the following components:
DME-Interface and subsystems

Application

Monitor

Diagnostics



2.4 DME-Interface and Subsystems

The DME-Interface is a PC based or „Black Box“ based piece of hardware that connects to all subsystems (Picture 2).

We will call the piece of software that runs within the DME-Interface”.

The driver handles all subsystems and provides TCP/IP sockets for communication.

When the DME is powered up, the driver will create up to 4 TCP/IP ports

Monitor port	(optional)
Diagnostics port	(optional)
Application port	(required)
Info port	(optional in V.1.0 will be required in future version)

2.4.1 Monitor

The machine monitor (monitor) is a piece of software that is used to display controller specific information like current machine position, active probe, ...

It connects to the monitor port to receive the displayed information from the DME-Interface.

The monitor is an optional component.

The controller may implement an equivalent functionality, for examples by displaying the machine position on the jog box display.

In most cases the DME vendor will supply the monitor.

A description of the monitor is not part of this specification.

2.4.2 Diagnostics

The machine diagnostics (diagnostics) is a piece of software that is used to display diagnostic information necessary to service, repair or setup the DME.

It connects to the diagnostic port to receive information from the DME-Interface.

The diagnostic is an optional component.

The DME vendor supplies the diagnostics.

A description of the diagnostics is not part of this specification.

2.4.3 Application

The application is a piece of software that runs on the client computer and that uses the application port to run the DME.

This specification describes the protocol used on the application port.

2.4.4 Info

The info is a piece of software that runs on the client computer and that uses the info port to provide information about the machine parameters (axis, sensors,...)

This specification describes the protocol used on the info port.

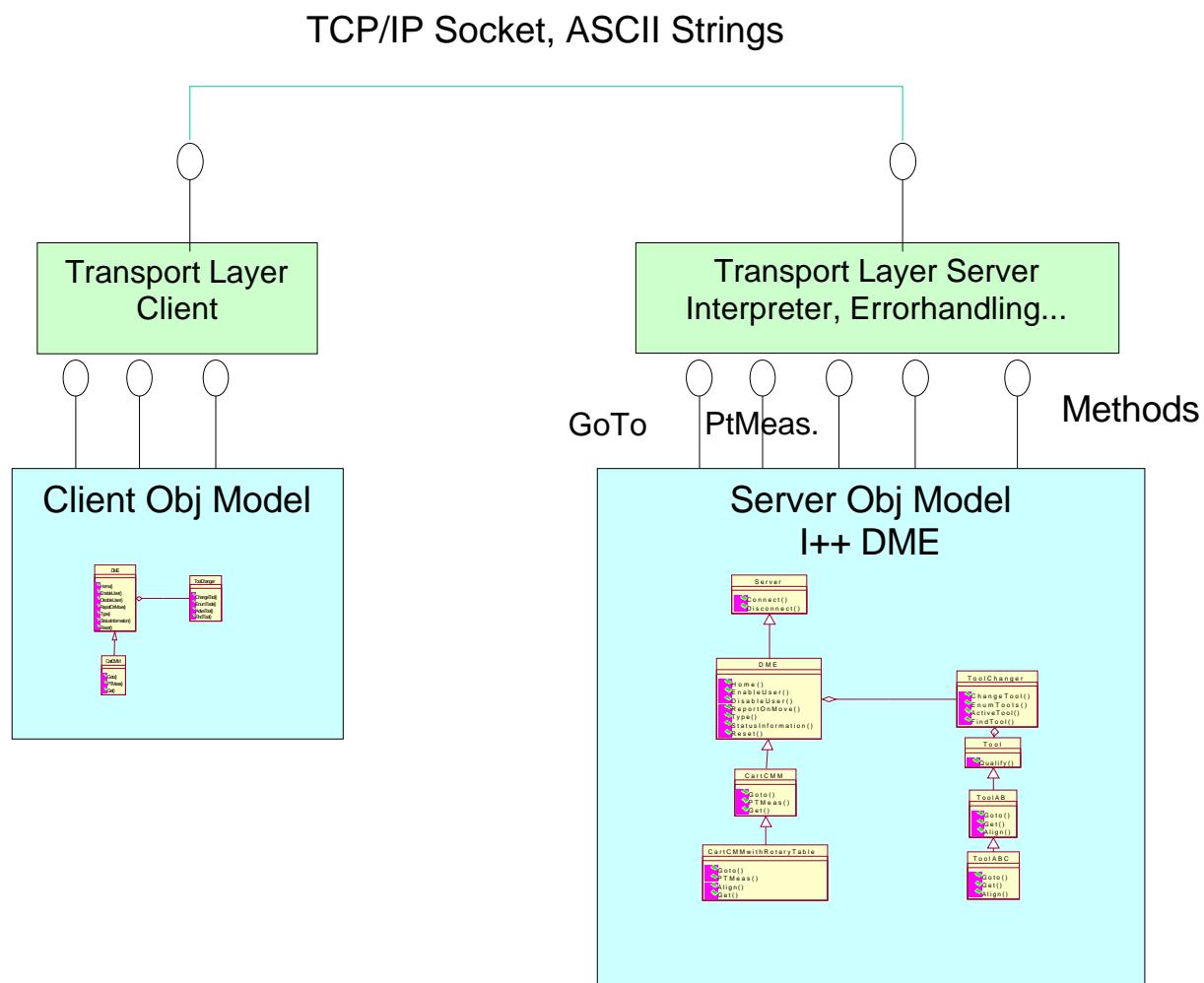
3 Hierarchy of Communication

3.1 Layers

The properties of the measuring equipment and the methods to handle them are defined by the object model, see picture 4.

The actual defined transport layer is to transmit ASCII strings via TCP/IP socket.

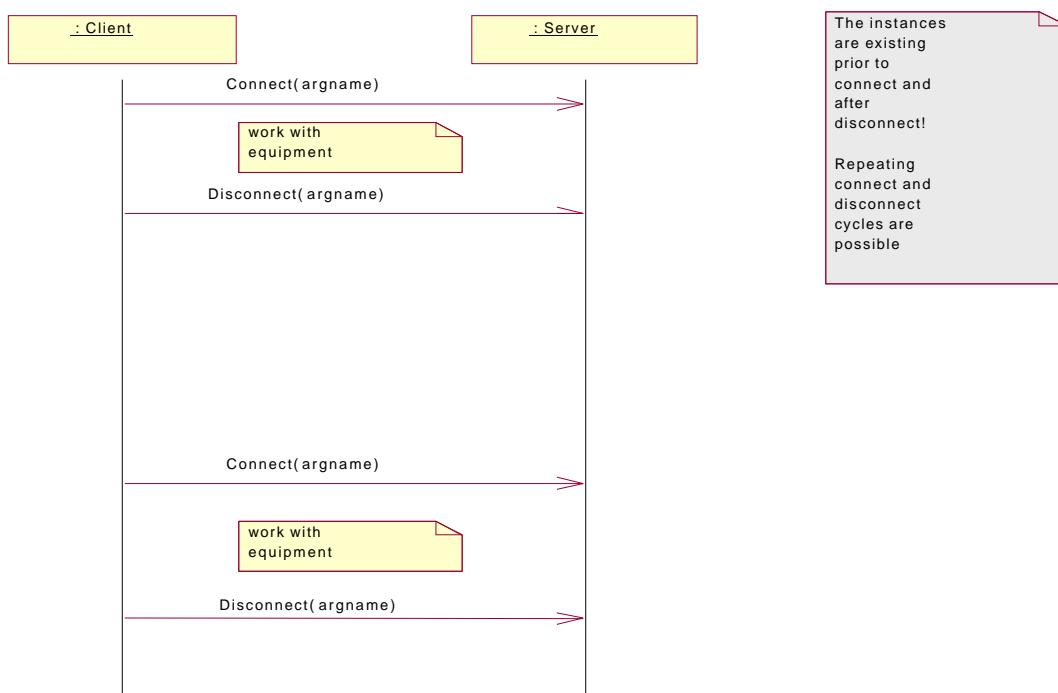
The layers are separated to have the chance to change the transport layer to future technologies.



3.2 Examples of basic use cases

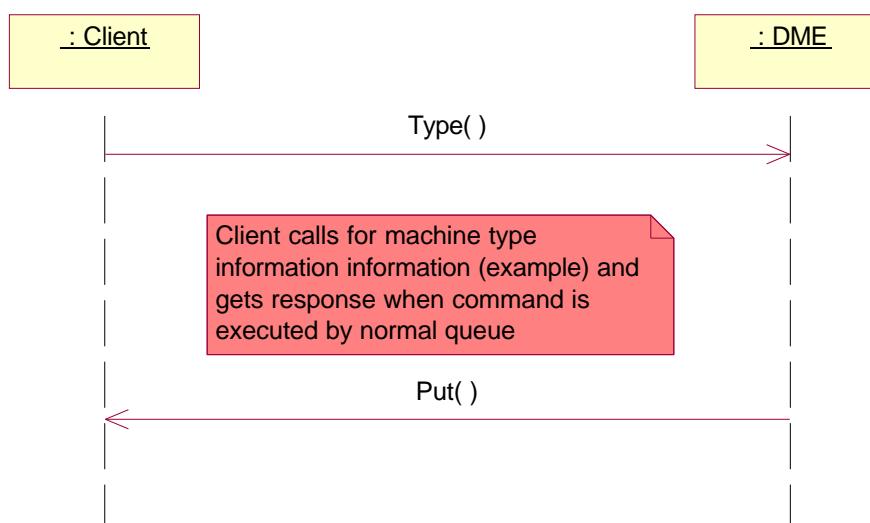
3.2.1 Sequence Diagram: Connect, Disconnect (Session)

Picture 5



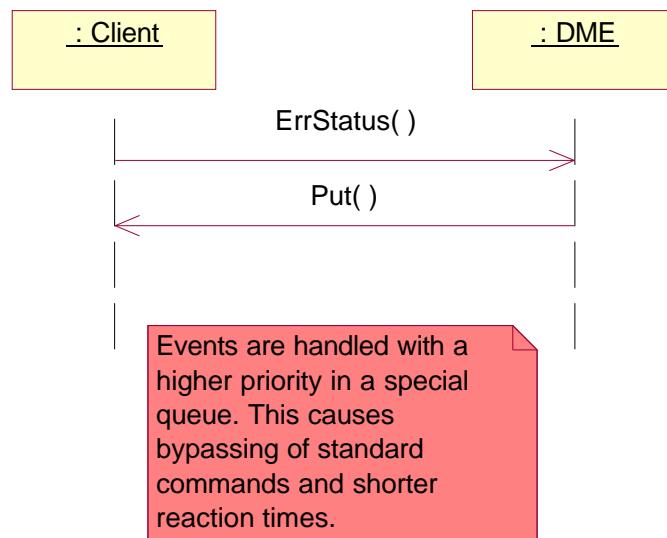
3.2.2 Sequence Diagram: Synchronous Communication

Picture 6



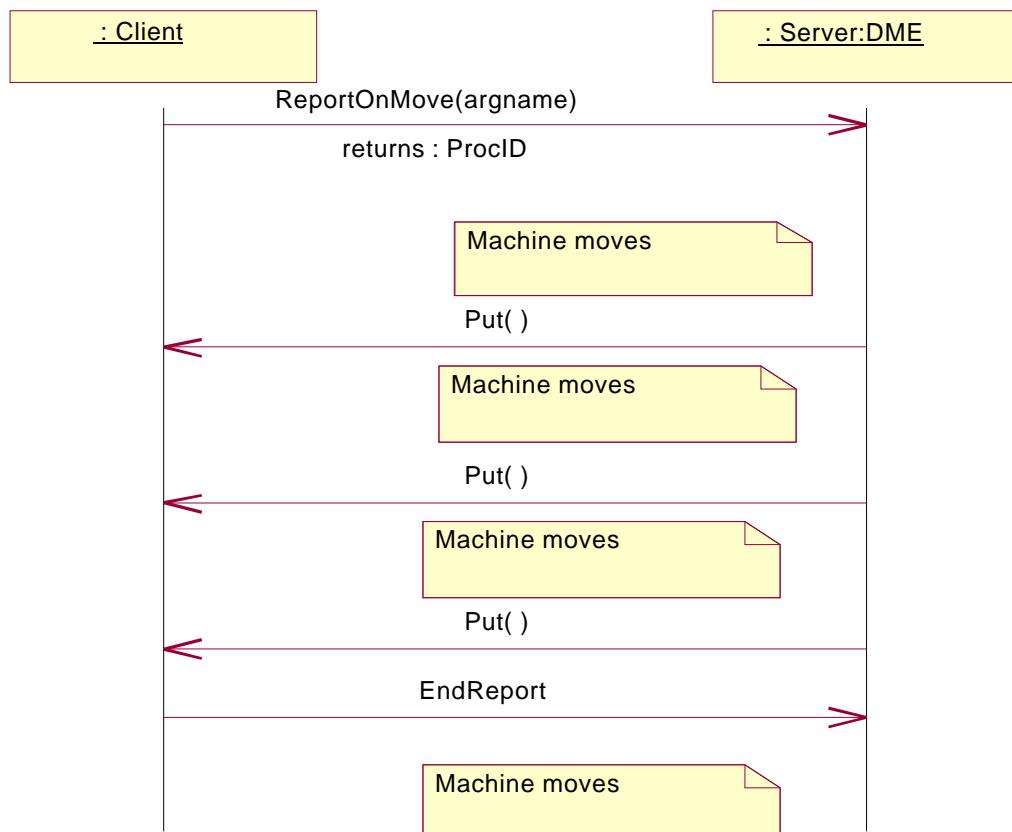
3.2.3 Sequence Diagram: Asynchronous Communication (Single Shot Events)

Picture 7



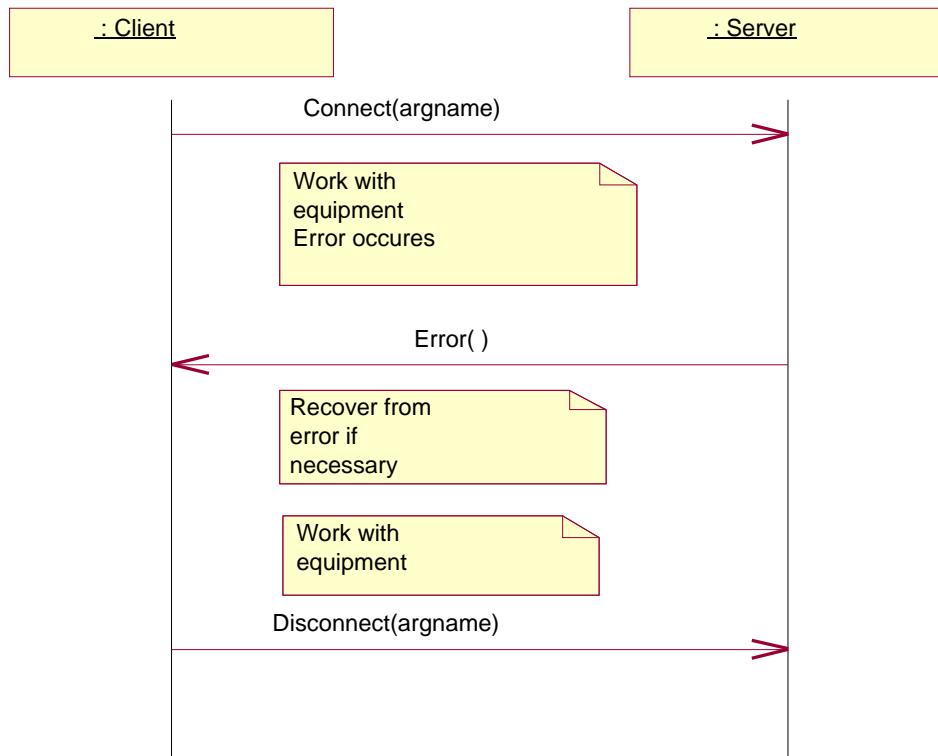
3.2.4 Sequence Diagram: Asynchronous Communication (Multiple Shot Events)

Picture 8



3.2.5 Sequence Diagram: Handling of Unsolicited Errors

Picture 9



4 Events

To increase performance and to reduce traffic on the interface the Event transactions are created. Events use tags starting with E.

4.1 Transaction events, syntax

Event transactions are initiated by the client.

Event requests are handled by the server with a higher priority as the synchronous communication. This means that the requests can bypass the normal command queue in the server.

In addition to normal transaction processing, the server will trigger an event. Legal tags are tags starting with E0001 up to E9999. The tag E0000 is reserved for events created by the server.

4.2 One shot events

These Events are used to generate exactly one asynchronous reaction of the server. F.I. getting asynchronous status or position information.

The transaction creates a daemon that triggers an event. The daemon will die after firing the event.

4.3 Multiple shot events

The transaction creates a daemon that triggers events based on a condition. The client must stop this daemons explicitly by a StopDeamon („Event transaction tag“) method.

4.4 Server events

Server events use tag E0000. They are used to report manual hits, key strokes, supported machine status changes...

5 Object Model

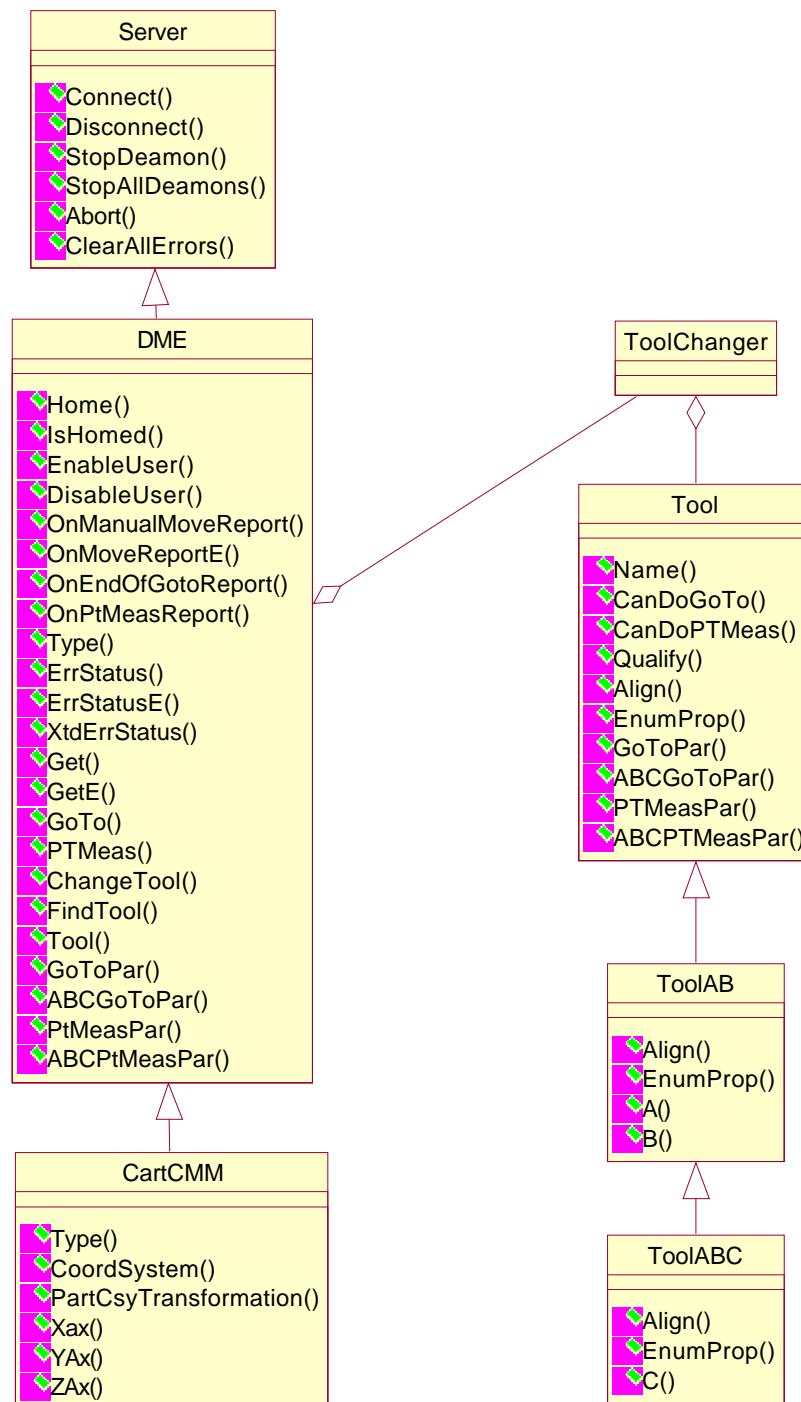
5.1 Explanation

The following diagrams (picture 10 and 11) show the designed class structure of the interface. It shows

- the classes representing the main components of a real coordinate measurement equipment (in this, first case coordinate measurement machine)
- the organization of methods and properties in these classes
- the relations between the main classes, the generalizations (specialization vice versa), the aggregations...
- this object model defines the structure of the interface and the syntax. It defines how to set and get the properties of the virtual components of this machine (chapter 6)
- Picture 10 is generated to help at a first step with the most important commands.
- Picture 11 is reengineered from and consistent to the header files (chapter 9). It shows also programming aspects as virtual definitions of methods in upper classes as DME and also property aspects as GoToPars and PTMeasPars blocks.

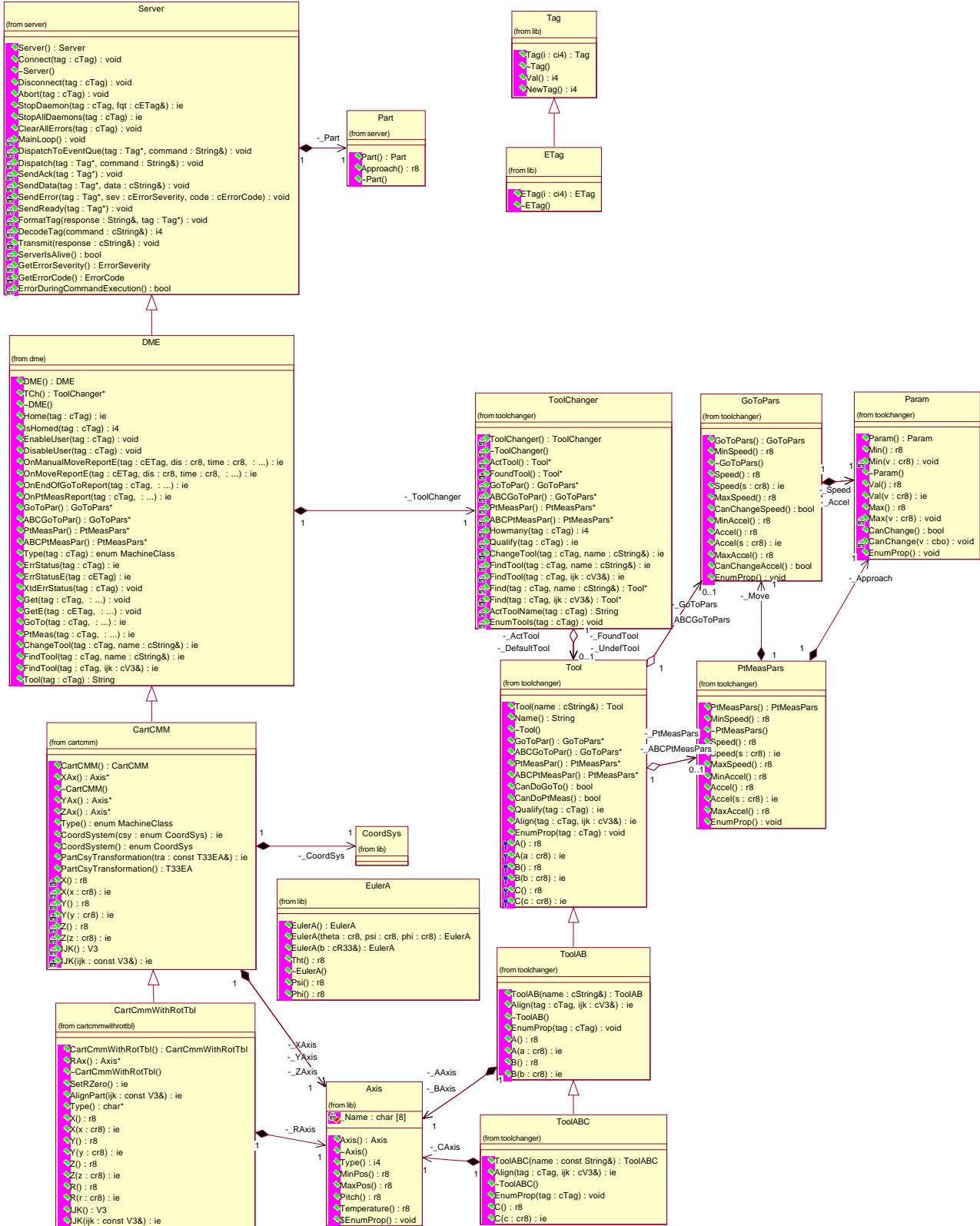
5.2 Reduced Object Model

Picture 10, basics, outside view, method oriented

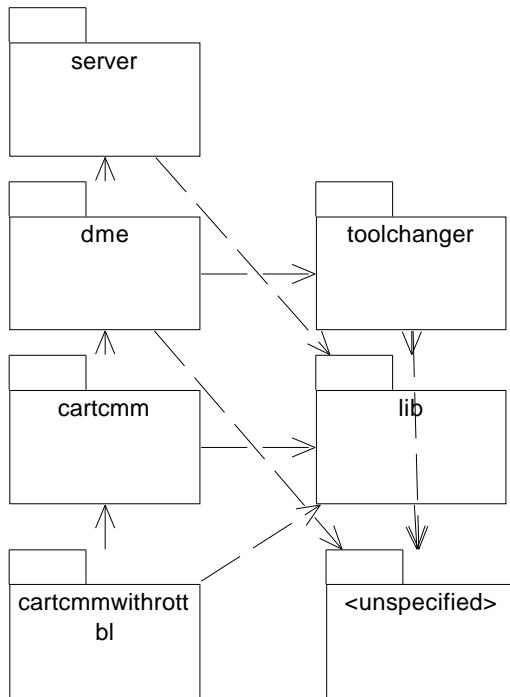


5.3 Full Object Model

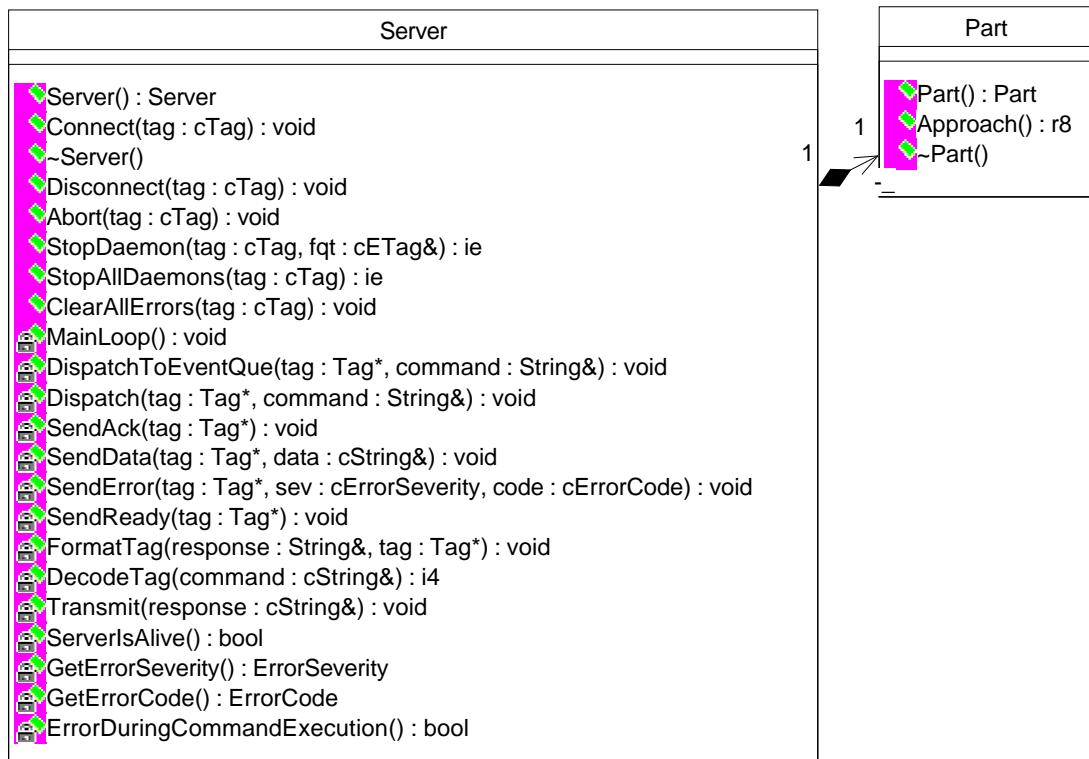
Picture 11, please zoom the .pdf file view.



5.4 Packaging for visualization, see header files in subdirectories chapter 9



5.5 Contents of server



5.6 Contents of dme

DME
<ul style="list-style-type: none"> ◆ DME() : DME ◆ TCh() : ToolChanger* ◆ ~DME() ◆ Home(tag : cTag) : ie ◆ IsHomed(tag : cTag) : i4 ◆ EnableUser(tag : cTag) : void ◆ DisableUser(tag : cTag) : void ◆ OnManualMoveReportE(tag : cETag, dis : cr8, time : cr8, : ...) : ie ◆ OnMoveReportE(tag : cETag, dis : cr8, time : cr8, : ...) : ie ◆ OnEndOfGoToReport(tag : cTag, : ...) : ie ◆ OnPtMeasReport(tag : cTag, : ...) : ie ◆ GoToPar() : GoToPars* ◆ ABCGoToPar() : GoToPars* ◆ PtMeasPar() : PtMeasPars* ◆ ABCPtMeasPar() : PtMeasPars* ◆ Type(tag : cTag) : enum MachineClass ◆ ErrStatus(tag : cTag) : ie ◆ ErrStatusE(tag : cETag) : ie ◆ XtdErrStatus(tag : cTag) : void ◆ Get(tag : cTag, : ...) : void ◆ GetE(tag : cETag, : ...) : void ◆ GoTo(tag : cTag, : ...) : ie ◆ PtMeas(tag : cTag, : ...) : ie ◆ ChangeTool(tag : cTag, name : cString&) : ie ◆ FindTool(tag : cTag, name : cString&) : ie ◆ FindTool(tag : cTag, ijk : cV3&) : ie ◆ Tool(tag : cTag) : String

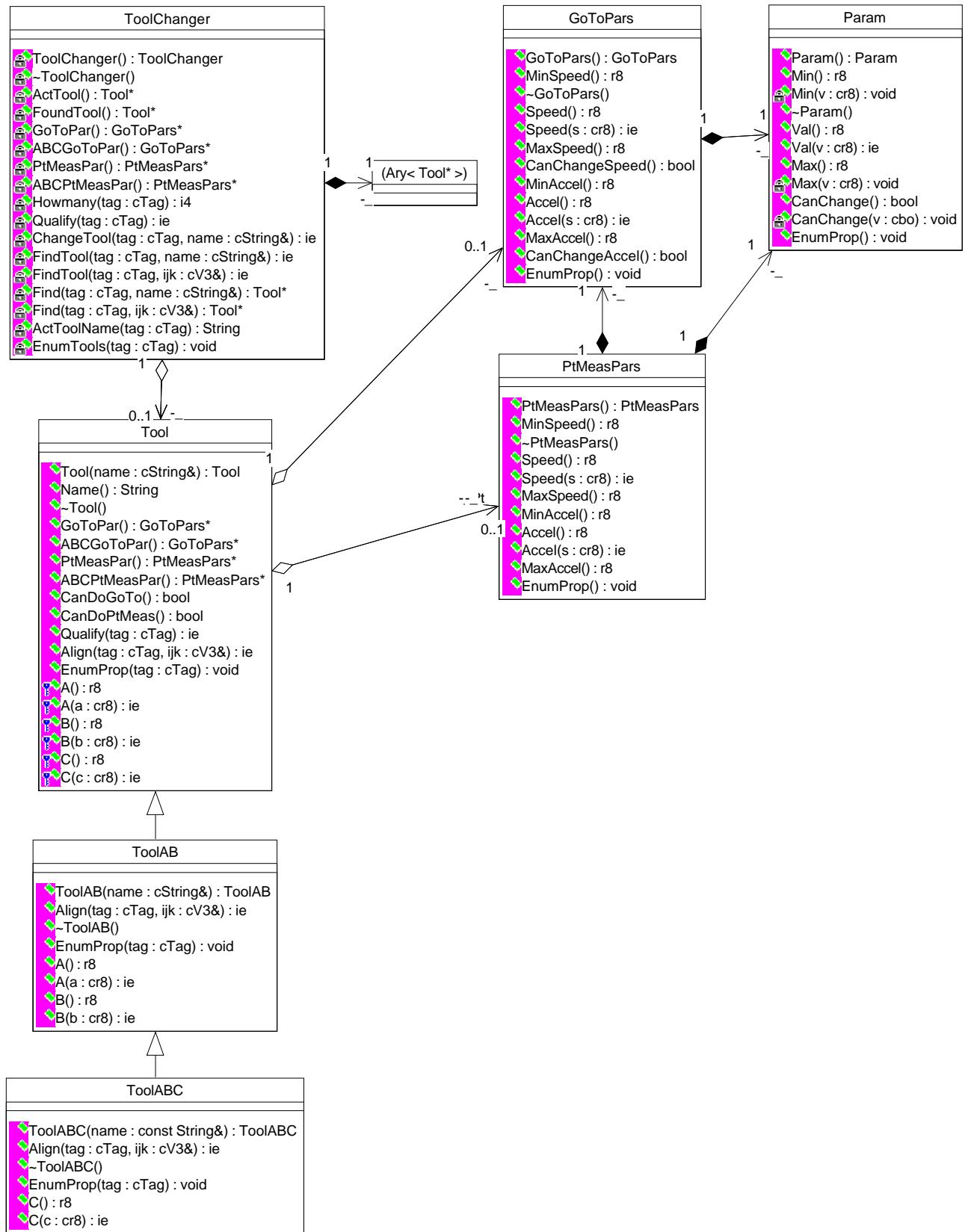
5.7 Contents of cartcmm

CartCMM
<ul style="list-style-type: none"> ◆ CartCMM() : CartCMM ◆ XAx() : Axis* ◆ ~CartCMM() ◆ YAx() : Axis* ◆ ZAx() : Axis* ◆ Type() : enum MachineClass ◆ CoordSystem(csy : enum CoordSys) : ie ◆ CoordSystem() : enum CoordSys ◆ PartCsyTransformation(tra : const T33EA&) : ie ◆ PartCsyTransformation() : T33EA ◆ X() : r8 ◆ X(x : cr8) : ie ◆ Y() : r8 ◆ Y(y : cr8) : ie ◆ Z() : r8 ◆ Z(z : cr8) : ie ◆ IJK() : V3 ◆ IJK(ijk : const V3&) : ie

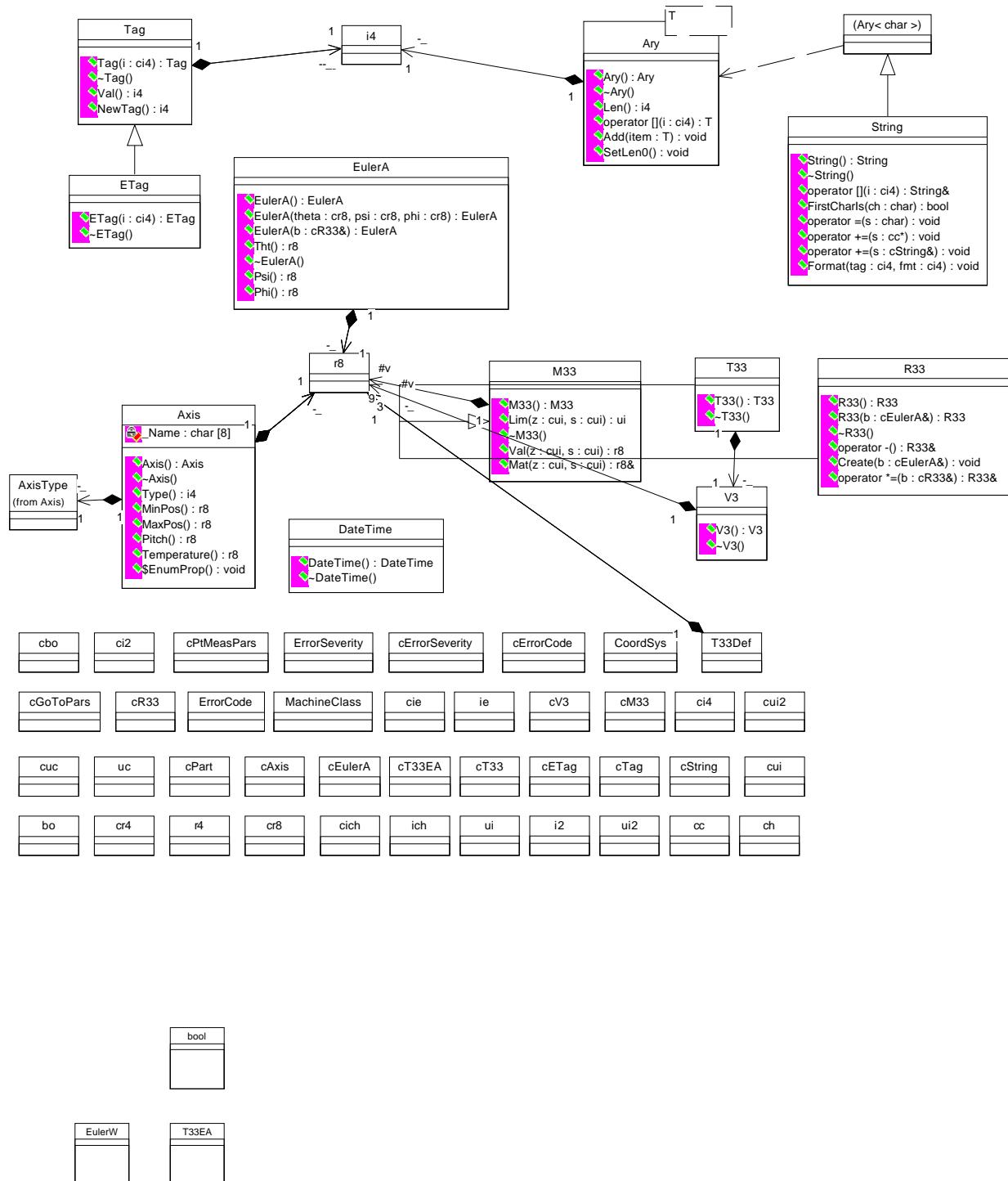
5.8 Contents of cartcmmwithrotarytable

CartCmmWithRotTbl
CartCmmWithRotTbl() : CartCmmWithRotTbl
RAx() : Axis*
~CartCmmWithRotTbl()
SetRZero() : ie
AlignPart(ijk : const V3&) : ie
Type() : char*
X() : r8
X(x : cr8) : ie
Y() : r8
Y(y : cr8) : ie
Z() : r8
Z(z : cr8) : ie
R() : r8
R(r : cr8) : ie
IJK() : V3
IJK(ijk : const V3&) : ie

5.9 Contents of toolchanger



5.10 Contents of lib and unspecified



6 Protocol

6.1 Communication

Communication between application client and I++DME server is based on the standard TCP/IP protocol.

6.1.1 Character set

All bytes received and send on the application port of the server are interpreted as 7 bit ASCII characters.

Only characters in the range from ASCII code = 40 (space) to ASCII code = 126 (~) can be used.

In addition the character pair Carriage Return (<CR>, ASCII code = 13) and Line Feed (<LF>, ASCII code = 10) is used as line terminator

<CR> and <LF> must always be send as a pair in the order <CR> followed by <LF>.

6.1.2 Initialization

After CMM power up the server will create the application port in listen mode.

When the client is started, it will send a connection request to the application port created by the server. The server will confirm the connection and is now ready to work with the client.

6.1.3 Shutdown

When the client no longer needs the server it will close the connection.

The driver will then listen on the application port for new incoming connection requests.

6.2 Protocol Basics

- Ø The protocol is line oriented.
- Ø Each line must be terminated by <CR><LF>.
- Ø The maximum number of characters in a line should not exceed 65535.
- Ø A line send from the client to the server is called CommandLine.
- Ø A line send from the server to the client is called ResponseLine.

In examples the terminating <CR><LF> is not shown !

6.2.1 Tags

The first 5 characters of each CommandLine represent a tag.

The client generates these tags. The client uses two types of tags:

- Ø CommandTag
- Ø EventTag

A CommandTag is a 5 digit decimal number with leading zeros present.

The number must be between 00001 and 09999.

The client is responsible for incrementing the tag number each time it sends a command.

Examples for tag created by the client:

```
04711      // tag is ok
01710      // ok
00020      // ok
20          // error; only 2 digits
00000      // error; out of range must be >=1 and <=9999
```

An EventTag is a 4 digit decimal number that is preceded by the character E(ASCII code=5).

The number must be between 0001 and 9999.

The client must use the same tag counter for CommandTag and EventTag.

The client is responsible for incrementing the tag counter each time it sends a command.

Examples for tag created by the client:

```
E3333      // tag is ok
E0456      // ok
E0000      // error; out of range must be >=1 and <=9999
E20        // error; only 3 characters
A4711      // error; illegal first character
```

As for a CommandLine the first 5 characters of a ResponseLine represent a tag (ResponseTag).

During normal command processing by the server it will use the tag received from the client as ResponseTag so the client can use this tag to relate the ResponseLine to a CommandLine.

In addition the server can send a ResponseLine using ResponseTag E0000 for reporting unsolicited events to the client.

6.2.2 General line layout

From now on we will use

- Ø Command as a synonym for CommandLine
- Ø Response as a synonym for ResponseLine.

6.2.2.1 CommandLine

The first 5 characters in each command line represent the CommandTag.

Character at column 6 must be a space (ASCII code = 40).

The command starts at column 7.

6.2.2.2 ResponseLine

The first 5 characters in each Response line represent the ResponseTag.

Character at column 6 must be a space (ASCII code = 40).

Character at column 7 can be one of the following:

- Ø &
- Ø %
- Ø !
- Ø #

The meaning of character at column 7 is explained later.

Character at column 6 must be a space when the line length is greater than 7.

6.2.2.3 Definitions

In the following we will use

- Ø Ready as a synonym for a ResponseLine where the 7th character is a %
- Ø Ack as a synonym for a ResponseLine where the 7th character is a &
- Ø Error as a synonym for a ResponseLine where the 7th character is a !
- Ø Data as a synonym for a ResponseLine where the 7th character is a #

6.2.3 Transactions

The basic protocol unit is a transaction. For each transaction, the client will create a tag. The tag identifies the transaction.

- Ø Transactions are initiated by the client.
- Ø The same tag is used during a transaction.
- Ø Transaction can overlap, which means, that a client can start a new transaction after having received an Ack from the server.
- Ø When using overlapped transaction, tags send to the server must be unique.
- Ø When using overlapped transactions and the server is too busy to accept new transactions it must delay sending the Ack until it is ready to accept a new transaction.
- Ø When using overlapped transaction, the server must make sure, that the Ack, Data (Error) and Ready are send back to the client in the right order. This means, if transaction 00001 is started before transaction 00002 the server is not allowed to send a Data, Error or Ready from transaction 00002 before the Ready from transaction 00001. At any point in time the server is allowed to send a line starting with an EventTag.

A transaction is complete after the server sends the Ready. If a transaction is complete, all processing on the driver side related to the transaction has completed.

6.2.3.1 Example

Client to Server	Server to Client	Comment
00001 Home()		Use tag 00001 for home command, client sends „home” command
	00001 &	Drv accepts command (Ack)
	00001 %	Drv reports to be ready (Ready)
00002 GoTo(X(100))		Move to x=100
	00002 &	command accepted (Ack)
	00002 %	position reached (Ready)
00003 GoTo(X(100000))		moving out of limits
	00003 &	command accepted
	00003 ! Error(2, 1, GoTo, Move Out of limits)	Error message
	00003 %	transaction complete
00004 Get(X(), Y())		get position of x, y axis
	00004 &	
	00004 # X(99.93), Y(17.148)	x and y position
	00004 %	transaction complete

6.2.4 Events

At any point in time the server may notify that something happened by sending an event to the client.

If the event is triggered by a transaction, the tag used is that of the transaction.

The server must first send an Ack before it can send the Response with the EventTag.

This Response can then be send before or after the Ready for the transaction.

At any point in time the server can send a Response with EventTag E0000 to inform the client that something unsolicited has happened in the server.

6.2.4.1 Examples

Unsolicited error message

Client to Server	Server to Client	Comment
	E0000 ! Error(3, 9, HealthCheck, Emergency button activated)	A unsolicited error message occurs
		In this example the server must display error and inform user what to do

Assume the user moves the machine using joysticks and the server wants to report this movement

Client to Server	Server to Client	Comment
00047 OnManualMoveReport(X(10), Y(10), Z(10))		
	00047 &	
	00047 %	
00048 EnableUser()		
	00048 &	
	00048 %	
E0553 OnManualMoveReportE(Time(1 ,dis(20),X()Y()Z())		
	E0553 &	
	E0553 %	
		Now the user moves the machine
	E0553 # X(50), Y(433), Z(500)	
	E0553 # X(50), Y(433), Z(520)	

	...	
00049 StopDaemon(E553)		The server now want to stop reporting and sends
	00049 &	
	E0553 # X(50), Y(433), Z(530)	
	00049 %	no events with tag E0553 may follow

6.2.5 Errors

If the server detects an error condition it will report the error using the tag of the Command it was executing when the error was detected.

The server will abort all pending transactions.

The client must invoke the ClearAllErrors() method before the server can continue processing commands.

6.3 Method Syntax

The reference for this description is the C++ class definition that is part of this documentation. Please note, that in the class description the first argument of all methods is Tag. This argument is converted into a CommandTag or EventTag as described before and therefore not part of this documentation (see Server::FormatTag() method).

6.3.1 Server Methods

A session defines the time period after the client has established the connection to the server on TCP/IP level until the client closes the communication socket to the server.

6.3.1.1 Connect()

After having completed the connection between client and server on TCP/IP level the Connect method initiates the connection between client and server. The server can be sure that the client will invoke Connect() only once during a session.

\emptyset Connect()

Parameters	None
Data	None.
Error	None.

Remarks	The server may for example use this method to perform initial checks Like which tool is active, ...
---------	--

The method does not perform any initializations, which means that the server is in a state that was left after power up or in a state that was left over from the previous session. The client can be sure, that no events or daemons are pending from the session before.

6.3.1.2 Disconnect()

The client invokes this method to end a session between client and server.
The client must close the TCP/IP connection after the transaction is complete.

\emptyset Disconnect()

Parameters	The method has no parameters.
Data	None.
Error	No errors are returned.
Remarks	The method must make sure, that all daemons are stopped and no events are send after it completes.

6.3.1.3 StopDaemon()

The client invokes this method to stop a daemon identified by its EventTag.

\emptyset StopDaemon(EventTag)

Parameters	EventTag of daemon to be stopped
Data	None.
Error	ErrorDaemonDoesNotExist

6.3.1.4 StopAllDaemons()

The client invokes this method to stop all daemons.

\emptyset StopAllDaemons()

Parameters	None
Data	None.
Error	None
Remarks	The method must make sure, that all daemons are stopped and no events are send after it completes.

6.3.1.5 Abort()

The client invokes this method to abort all pending transactions.

\emptyset Abort()

Parameters	None
Data	None.
Error	None
Remarks	The client must invoke the ClearAllErrors() method before the server will process new methods.

6.3.1.6 ClearAllErrors()

The client invokes this method to enable the server to recover from an error

\emptyset ClearAllErrors()

Parameters	None
Data	None.
Error	None

Examples

Client to Server	Server to Client	Comment
00051 GoTo(X(1000)		
	00051 &	
00052 GoTo(Y(300)		
	00052 &	
		The client wants to abort the moves and sends
00053 Abort()		
	00053 &	

	00051 ! Error(..ErrorSTransaction Aborted)	
	00051 %	
	00052 ! Error(..ErrorSTransaction Aborted)	
	00052 %	
	00053 %	
		If the client now sends
00054 Get(X())		
	00054 &	
	00054 ! Error(..ErrorSProcessing Method)	
	00054 ! Error(..ErrorIUseClearAll ErrorsToContinue)	
	00054 %	the server will still be in a error state
00055 ClearAllErrors()		
	00055 &	
	00055 %	the server is now ready to accept new method calls
00056 Get(X())		
	00056 &	
	00056 # X(23)	
	00056 %	

6.3.2 DME Methods

6.3.2.1 Home()

The client uses this method to home the machine. The server must be homed before the client can invoke methods that move the machine.

\emptyset Home()

Parameters	None.
Data	None.
Errors	ErrorDuringHome

6.3.2.2 IsHomed()

The client uses this method to query if the machine is homed

\emptyset IsHomed()

Parameters	None.
Data	IsHomed(Bool).
	Bool = 0 not homed
	Bool = 1 is homed
Errors	None.

6.3.2.3 EnableUser()

The client uses this method to enable user interaction with the machine.

\emptyset EnableUser()

Parameters	None.
Data	None
Errors	None.

Remarks	The method will have arguments in the next version to allow the client to enable only a subset of the user interface elements like specific keys or joysticks only.
---------	---

6.3.2.4 DisableUser()

The client uses this method to disable user interaction with the machine.

\emptyset DisableUser()

Parameters	None.
Data	None
Errors	None.

Remarks	The server calls this method implicitly whenever the client calls a method that physically moves the machine.
---------	---

6.3.2.5 IsUserEnabled()

The client uses this method to query if the user is enabled.

\emptyset IsUserEnabled()

Parameters	None.
Data	IsUserEnabled(Bool). Bool = 0 user is disabled Bool = 1 user is enabled
Errors	None.
Remarks	The client should check if the user is enabled after each Connect() and not rely on a default.

6.3.2.6 OnEndOfGoToReport()

The client uses this method to define which information the server should send to the client when the server is has completed the GoTo command.

\emptyset OnEndOfGoToReport ()

Parameters	Enumeration of methods that return a property. If the enumeration is empty, no reports are send.
Data	None.
Errors	ErrorBadProperty .
Remarks	The server will send a report after the GoTo commands has completed.

6.3.2.7 OnPtMeasReport()

The client uses this method to define which information the server should send to the client when the server is has completed the PtMeas command.

\emptyset OnPtMeasReport ()

Parameters	Enumeration of methods that return a property. The enumeration may not be empty.
Data	None.
Errors	ErrorBadProperty .
Remarks	The server will send a report after the PtMeas commands has completed.

6.3.2.8 OnMoveReportE()

The client uses this method to define which information the server should send to the client while the server is executing GoTo commands.

\emptyset OnMoveReportE ()

Parameters	Time(s), Dis(d), ... Enumeration of methods that return a property. If the enumeration is empty, no reports are send.
Data	None.
Errors	ErrorBadProperty .
Remarks	The server will send a report if the time interval s has elapsed and the machine has moved more than d millimeters. The report frequency must not be higher than 10 Hz.

Example OnMoveReportE(Time(0.5), Dis(0.2), X(), Y(), Z())

6.3.2.9 OnManualMoveReportE()

This method is identical to OnMoveReportE() when the UserIsEnabled and moves the machine by joysticks.

\emptyset OnManualMoveReportE ()

Parameters	Time(s), Dis(d), ... Enumeration of methods that return a property. If the enumeration is empty, no reports are send.
Data	None.
Errors	ErrorBadProperty .
Remarks	The server will send a report if the time interval s has elapsed and the machine has moved more than d millimeters. The report frequency must not be higher than 10 Hz.

Example OnManualMoveReportE(Time(0.5), Dis(0.2), X(), Y(), Z())

6.3.2.10 Type()

The client uses this method to query the type of machine.

\emptyset Type()

Parameters	None.
Data	One of the following must be returned: Type(CartCMM)
Errors	None .

6.3.2.11 ErrStatus()

The client uses this method to query the error status of server.

\emptyset ErrStatus()

Parameters	None.
Data	ErrStatus(Bool).

	Bool	= 1	in error
	Bool	= 0	ok
Errors	None .		

6.3.2.12 ErrStatusE()

The client uses this method to query the error status of server.

\emptyset ErrStatusE()

Parameters	None.		
Data	ErrStatus(Bool).		
	Bool = 1	in error	
	Bool = 0	ok	
Errors	None .		

6.3.2.13 XtdErrStatus()

The client uses this method to query the extended error status of server.

\emptyset XtdErrStatus()

Parameters	None.
Data	The server may send one or more lines of status information like
	IsHomed(1)
	IsUserEnabled(0)
	...
Errors	as well as one or more Errors like
	Error(ErrorIAirPressureInRange)
	Error(ErrorINoDaemonsAreActive)

6.3.3 CartCMM Methods

Each CartCMM implements a cartesian machine coordinate system.
Based on this coordinate system the following depend on it:

- MachineCsy
- MultipleArmCsy
- PartCsy

The CMM will for some commands use execution parameters that are defined in the context of a tool like speed and acceleration for executing a GoTo command.

6.3.3.1 CoordSystem()

The client uses this method to select the coordinate system it wants to work with.

\emptyset CoordSystem(Param)

Parameters	One of the following: MachineCsy MultipleArmCsy PartCsy
Data	None.
Errors	ErrorBadParameter

6.3.3.2 CoordSystem()

The client uses this method to query the server which coordinate system is selected..

\emptyset CoordSystem()

Parameters	None.
Data	CoordSystem(Arg)
	Arg can be one of the following: MachineCsy MultipleArmCsy PartCsy
Errors	None.

6.3.3.3 PartCsyTransformation()

The client uses this method to get the part coordinate transformation back from the server.

\emptyset PartCsyTransformation()

Parameters	None.
Data	PartCsyTransformation(X0, Y0, Z0, Theta, Psi, Phi)
Errors	None.
Remarks	Definition of relation between transformation matrix and parameters is given in C++ class definition.

6.3.3.4 PartCsyTransformation(..)

The client uses this method to define the part coordinate transformation to transform machine coordinates into part coordinates and vice versa.

\emptyset PartTransformation(X0,Y0,Z0, Theta, Psi, Phi)

Parameters	X0,Y0, Z0 define the zero point of the machine coordinate system in part coordinates. Theta, Psi and Phi are Euler angles that define the rotation matrix of the transformation.
Data	None
Errors	ErrorThetaOutOfRange

6.3.3.5 X()

\emptyset X()

Parameters	None.
Data	X(x)
Errors	ErrorUndefTool ErrorBadContext
Remarks	This method can only be invoked as argument of a Get or OnReport method.

6.3.3.6 Y()

\emptyset Y()

Parameters	None.
Data	Y(y)
Errors	ErrorUndefTool ErrorBadContext
Remarks	This method can only be invoked as argument of a Get or OnReport method.

6.3.3.7 Z()

\emptyset Z()

Parameters	None.
Data	Z(z)
Errors	ErrorUndefTool ErrorBadContext
Remarks	This method can only be invoked as argument of a Get or OnReport method.

6.3.3.8 IJK()

\emptyset IJK()

Parameters	None.
Data	IJK(I,j,k)
Errors	ErrorBadContext.
Remarks	i,j,k define a direction vector in the machine coordinate system. The vector is not necessarily normalized. Its values are tool dependent. If the client normalizes the vector it should point out of the part material. This method can only be invoked as an argument of OnPtMeasReport().

6.3.3.9 X(..)

\emptyset X(x)

Parameters	target x position
Data	None. ...
Errors	ErrorMoveOutOfLimits ErrorCollision ErrorBadContext.
Implicit	Tool()->GoToPars
Remarks	This method can only be invoked as argument of a GoTo or PtMeas method

If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.10 Y(..)

\emptyset Y(y)

Parameters	target y position
Data	None. ...
Errors	ErrorMoveOutOfLimits ErrorCollision ErrorBadContext.
Implicit	Tool()->GoToPars
Remarks	This method can only be invoked as argument of a GoTo or PtMeas method If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.11 Z(..)

\emptyset Z(z)

Parameters	target z position
Data	None. ...
Errors	ErrorMoveOutOfLimits ErrorCollision ErrorBadContext.
Implicit	Tool()->GoToPars
Remarks	This method can only be invoked as argument of a GoTo or PtMeas method If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.12 IJK(..)

\emptyset IJK(I,j,k)

Parameters	I,j,k define the X,Y,Z values of a vector.
Data	None
Errors	ErrorBadContext. ErrorVectorHasNoNorm ErrorBadContext.
Remarks	i,j,k define a direction vector in the machine coordinate system. The vector is not necessarily normalized. Before using the vector, the server must normalize it. This method can only be invoked as an argument of another method.

6.3.3.13 Get()

The client uses this methods to query the position of the active tool.

\emptyset Get()

Parameters	An enumeration of one or more of the following methods can be argument.
------------	---

X()
Y()
Z()

or other methods that return properties.

Data	The format is defined by the method enumerated.
Errors	Errors of the enumerated methods.

6.3.3.14 GetE()

The client uses this methods to query the position of the active tool.

\emptyset Get()

Parameters	An enumeration of one or more of the following methods can be argument.
	X() Y() Z()

or other methods that return properties.

Data	The format is defined by the method enumerated.
Errors	Errors of the enumerated methods.

6.3.3.15 GoTo()

The client uses this method to perform a multi axis move to the target position using the active tool.

\emptyset GoTo(..)

Parameters	An enumeration of one or more of the following methods can be argument.
	X(..) Y(..) Z(..)

Or methods that change tool properties (like A, B).

Data	As defined by OnEndOfGoToReport() method.
Errors	Errors of the enumerated methods.
Implicit	Tool()->GoToPars

Remarks

The server will move the machine, so that all axes enumerated will start to move and stop to move at the same time. The tool moves on a straight line in the MachineCsy.

6.3.3.16 PtMeas()

The client uses this method to execute a single point measurement using the active tool.
Parameters necessary (Speed, clearance distance, ..) are defined by the active tool

\emptyset PtMeas(..)

Parameters	An enumeration of one or more of the following methods can be argument.
	X(..)
	Y(..)
	Z(..)
	IJK(..)
Data	As defined by OnPtMeasReport() method
Errors	ErrorSurfaceNotFound.
Implicit	Tool()->PtMeasPar
	Tool()->GoToPar

Remarks

The PtMeas() method is processed by the server as follows:

If a IJK vector is present

- § The vector I, J, K is normalized
- § The position X, Y, Z is shifted into I, J, K direction by the sum of following values:
 - Part()->Clearance()
 - Tool()->Approach()
 - Tool()->Radius()

This new position is called approach position.

- § The server moves the machine to the approach position. This move is executed like an implicit GoTo().
- § The position X, Y, Z is shifted in the opposite I, J, K direction by the value of Tool()->Search(). This position is called end of search position.
- § The server moves the machine to end of search position using the PTMeasPars of the Tool().
- § If the tool has part contact during this move the server latches the position of the center of the ActiveTool and reports to the client as defined by OnPtMeasReport().
- § After contact the server will move the machine according to the value of Tool().Retract() using Tool().GoToPars for the move. If Tool().Retract() is greater or equal zero, the server will shift the contact position into IJK direction by this value and move the machine to this position. If the Tool().Retract() is less than zero, the server will move back to the approach position as defined before.

If a IJK vector is not present

- § The position before invocation of the method is the approach position.
- § The values of X, Y, Z define the end of search position.
- § The direction from the end of search position to the approach position implicitly define a IJK direction.
- § The server moves the machine to end of search position using the Tool()->PtMeasPars for speed and acceleration.
- § If the tool has part contact during this move the server latches the position of the center of the ActiveTool and reports to the client as defined by OnPtMeasReport().

§ After contact the server will move the machine according to the value of Tool().GetRetract() using Tool().GoToPars. If this value greater or equal zero, the server will shift the contact position into IJK direction by this value and move the machine to this position.

If the Tool().GetRetract() is less than zero, the server will move back to the approach position as defined before.

Note that the end of search position may be outside the move limits, but the part surface inside. In this case the server will report success if the surface is still inside or ErrorMoveOutOfLimits. This behavior differs from the GoTo() method.

7 Additional Dialog Examples

7.1 Connect

Client to Server	Server to Client	Comment
		Server and Client must be boot up previously
00001 Connect		Client connects to server
	00001 &	Server sends acknowledge
	00001 %	Server sends ready

7.2 Move 1 axis

Client to Server	Server to Client	Comment
00009 PartCsyTransformation(10, 20,30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 CoordSystem(PartCsy)		Select transformation to and from part coordinate system
	00010 &	
	00010 %	
00011 GoTo(X(100))		Move now in part coordinate system
	00011 &	
	00011 %	

7.3 Probe 1 axis

Client to Server	Server to Client	Comment
00014 OnPTMeasReport(X(),Y(),Z(),Tool().A())		Client defines format for probing result. Valid for every PTMeas command from now on.
	00014 &	
	00014 %	
00015 PTMeas(X(200))		Uses standard method in CartCMM
	00015 &	
	00015 # X(199.998),Y(250.123),Z(300.002),Tool().A(45)	Probing result from server
	00015 %	

7.4 Move more axis in workpiece coordinate system

Client to Server	Server to Client	Comment
00009 PartCsyTransformation(10, 20,30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 CoordSystem(PartCsy)		Select transformation to part coordinate system
	00010 &	
	00010 %	
00011 Goto(X(100),Y(150),Z(200))		Move with more axis
00011 Goto(X(100),Y(150),Z(200),R(180))		Alternatively
	00011 &	
	00011 %	

7.5 Probe with more axis

Client to Server	Server to Client	Comment
00014 OnPTMeasReport(X(),Y(),Z(),Tool().A())		Valid for every PTMeas command
	00014 &	
	00014 %	
00015 PTMeas(X(200),Y(250),Z(300))		Uses standard method in CartCMM
00015 PTMeas(X(200),Y(250),Z(300), IJK(0,0,1))		Alternatively, with approaching vector
00015 PTMeas(X(200),Y(250),Z(300), IJK(0,0,1),R(180))		Alternatively, with approaching vector and rotary table
	00015 &	
	00015 # X(199.998),Y(250.123),Z (300.002),Tool().A(45)	Result
	00015 %	

7.6 Set property

Client to Server	Server to Client	Comment
00015 ActiveTool().PTMeasPars().SetSpeed(100)		Set probing speed of active tool
	00015 &	
	00015 %	
00015 FindTool(Probe1).PTMeasPars() .SetSpeed(100)		Set probing speed of Probe1
	00015 &	
	00015 %	

7.7 Get, read property

Client to Server	Server to Client	Comment
00014 Tool().PTMeasPars().EnumProp() ()		Get ActiveTools PTMeas Property list
	00014 &	
	00014 # Speed	
	00014 # Accel	
	
	00014 # Approach	
	00014 %	
00015 Tool().PTMeasPars().getspeed()		Request for getting probing speed of active tool
	00015 &	
	00015 # Speed(100)	
	00015 %	
00015 FindTool(Probe1)		Search pointer to Probe 1
	00015 &	
	00015 %	
00016 FindTool().PTMeasPars().GetSpeed()		Get Probing speed of Probe1
	00016 &	
	00016 # Speed(100)	
	00016 %	

8 Error Handling

- Each transaction can generate multiple error messages.
- These messages are headed by the same tag number.

8.1 Classification of Errors

0003 ! Error(F1, F2, F3, Text)

8.2 F1: Error severity classification

0: Info

1: Warning

2: Error, client should be able to repair the error

3: Error, user interaction necessary

9: Fatal server error

Errors with classification higher or equal 2 require ClearError().

F2: Error numbers, 0000-4999, defined by I++ DME

5000-9999 definable from server

F3: I++ recommends to serve here the name of the error causing method

8.3 List of I++ predefined errors

Classification in Field F2

0000-0499 Protocol, syntax error

0 0000: Buffer full

2 0001: Illegal Tag

2 0002: no Space at pos. 5

2 0003: Illegal Flag Character at pos. 6

2 0004: No Space at pos. 7

1 0005: Suspend Communication

0500-0999 Error generated during execution in DME (see object model)

1000-1499 Error generated during execution in CartCMM... (see object model)

1500-1999 Error generated during execution in ToolChanger (see object model)

2000-2499 Error generated during execution in Tool... (see object model)

2500-2999 Error generated during execution in Axis (see object model)

9 C++ and Header Files for Explanation

9.1 \main\main.cpp

```
-----  
#include "../cartcmm/cartcmm.h"  
//#include "../dme/dme.h"  
//#include "../cartcmmwithrottbl/CartCmmWithRotTbl.h"  
  
-----  
//Server      _Server;  
//Server*     Srv()    {return &_Server;}  
  
void main () {  
  
    //r8          speed = Srv()->GoToPar()->Speed();  
    //ie          r   = Srv()->GoToPar()->Speed(5.0);  
  
    -----  
};
```

9.2 \server

9.2.1 \server\server.h

```
#if !defined(AFX_Server_H_E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)  
# define AFX_Server_H_E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_  
  
#include "Part.h"  
#include <ETag.h>  
  
-----  
class Server {  
  
    Part      _Part;  
  
    -----  
public:      Server  ();  
virtual ~Server ();  
  
    -----  
void        Connect (cTag tag);  
void        Disconnect(cTag tag);  
void        Abort   (cTag tag);  
                                         // connect to client  
                                         // disconnect from client  
                                         // abort pending transactions  
  
    -----  
ie          StopDaemon (cTag tag, cETag &fqt);  
ie          StopAllDaemons(cTag tag);  
                                         // stop daemon  
                                         // stop all daemons  
  
    -----  
void        ClearAllErrors(cTag tag);  
  
    -----  
private:  
methods are for  
                                         // these  
  
                                         // documentation purpose only  
  
void        MainLoop          ();  
void        DispatchToEventQue (Tag *tag, String &command);  
void        Dispatch           (Tag *tag, String &command);  
void        SendAck            (Tag *tag);  
void        SendData           (Tag *tag, cString &response);  
void        SendError          (Tag *tag, cErrorSeverity sev, cErrorCode code);  
void        SendReady          (Tag *tag);  
void        FormatTag         (String &response, Tag* tag);  
i4          DecodeTag         (cString &command);  
void        Transmit           (cString &response);
```

```

//-----
bool      ServerIsAlive();
ErrorSeverity GetErrorSeverity();
ErrorCode    GetErrorCode();
bool       ErrorDuringCommandExecution();

//-----
};

#endif

```

9.2.2 \server\part.h

```

#ifndef AFX_Part_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_
#define AFX_Part_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include <lppTop.h>

//-----

class Part {

//-----

r8          _Approach;
r8          _XpanCoefficient;
r8          _Temperature;

//-----

public:      Part      ();
virtual     ~Part     ();

//-----

r8          Approach()   {return _Approach;}

//-----
};

#endif

```

9.2.3 \server\server.cpp

```

//-----

#include "String.h"
#include "Server.h"

//-----

void      Server::MainLoop()                                { // for documentation only ***
// this method is implemented for
// documentation purpose only
String      command;
Tag*        tag   = Nil;
do {
    // wait for a command line from client
    // .. Wait(command)
i4
2 to 5
    tagval = DecodeTag(command); // get tag values define by chars from
                                // receive
                                if (command.FirstChar('E')) {
                                tag = new ETag(tagval);
                                SendAck(tag); // confirm
                                DispatchToEventQue(tag, command); // do whatever is necessary
                                else {
                                tag = new Tag(tagval);

```

```

        SendAck(tag);                                // confirme
receive
        Dispatch(tag, command);                     // do whatever is necessary

        if (ErrorDuringCommandExecution()) {          // something went
wong ?
            SendError(tag, GetErrorSeverity(), GetErrorCode());} // send error message

            SendReady(tag);}                         // while server is alive
while (ServerIsAlive());}

//-----
void    Server::Transmit(cString &response)   {                                // for documentation only ***
// send this string to client
/*... Send(response) */}

//-----
void    Server::FormatTag(String &response, Tag* tag){ // for documentation only ***
remove all chars
    response.SetLen0();                           // remove all chars
leading zeros
    response.Format(tag->Val(), 5);             // format 5 digits with
if (dynamic_cast<ETag*>(tag) !=Nil) {
    response[1] = 'E';}
    response += " ";}                           // use E to indicate event
// append a space char

//-----
void    Server::SendAck(Tag* tag)   {           // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "&";
    Transmit(response);}                         // add %

//-----
void    Server::SendData(Tag *tag, cString &data) {           // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "#";                            // add # and
space
    response += data;                          // add data
    Transmit(response);}

//-----
void    Server::SendError(Tag *tag, cErrorSeverity sev, cErrorCode code) { // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "! ";
//    response += ...;
    Transmit(response);}                         // add ! and space
// add error

//-----
void    Server::SendReady(Tag *tag) {           // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "%";                           // add %
    Transmit(response);}

//-----
```

9.3 \dme

9.3.1 \dem\dme.h

```
#if !defined(AFX_DME_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
# define AFX_DME_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../server/Server.h"
#include "../toolchanger/ToolChanger.h"

//-----

class DME : public Server {

    ToolChanger           _ToolChanger;

//-----

    bool                  _IsHomed;
    bool                  _IsUserEnabled;

//-----

public:          DME      ();
virtual         ~DME     ();

//-----

    ToolChanger*          TCh()                      {return &_ToolChanger;}

//-----

    virtual   ie           Home  (cTag tag)           {};
    virtual   ie           i4        lsHomed(cTag tag)  {}           {return _IsHomed;};

//-----

    virtual   void          EnableUser (cTag tag)    {};
    virtual   void          DisableUser(cTag tag)    {};

//-----

    ie                     OnManualMoveReportE(cETag tag, cr8 dis, cr8 time, ...);
    ie                     OnMoveReportE   (cETag tag, cr8 dis, cr8 time,...);

//-----

    ie                     OnEndOfGoToReport (cTag tag, ...);
    ie                     OnPtMeasReport  (cTag tag, ...);

//-----

    GoToPars*             GoToPar ()           const {return _ToolChanger.GoToPar();}
    GoToPars*             ABCGoToPar()        const {return _ToolChanger.ABCGoToPar();}

//-----

    PtMeasPars*           PtMeasPar ()         const {return _ToolChanger.PtMeasPar();}
    PtMeasPars*           ABCPtMeasPar()       const {return _ToolChanger.ABCPtMeasPar();}

//-----

    virtual   MachineClass Type(cTag tag)           {return CMMIsUndefCMM;};

//-----

    ie                     ErrStatus (cTag tag);
    ie                     ErrStatusE  (cETag tag);
    void                  XtdErrStatus(cTag tag);

//-----

    void                  Get      (cTag   tag, ...);
    void                  GetE     (cETag  tag, ...);

//-----
```

```

ie GoTo          (cTag   tag,...);
ie PtMeas        (cTag   tag,...);

//-----
ie ChangeTool    (cTag tag, cString &name) {return TCh()->ChangeTool (tag,
name);}
ie FindTool     (cTag tag, cString &name) {return TCh()->FindTool  (tag, name);}
ie FindTool     (cTag tag, cV3 &ijk)      {return TCh()->FindTool  (tag, ijk);}
String          Tool      (cTag tag)       {return TCh()-
>ActToolName(tag);}

//-----
};

#endif

```

9.4 \cartcmm

9.4.1 \cartcmm\cartcmm.h

```

#ifndef !defined(AFX_CartCMM_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_CartCMM_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../DME/dme.h"
#include "T33.h"

//-----

class CartCMM :public DME {

Axis           _XAxis;
Axis           _YAxis;
Axis           _ZAxis;

//-----

CoordSys      _CoordSys;
T33           _PartCoordTrandformation;

//-----

public:         CartCMM ();
virtual        ~CartCMM () ;

//-----

Axis*          XAx()   {return &_XAxis;}
Axis*          YAx()   {return &_YAxis;}
Axis*          ZAx()   {return &_ZAxis;}

//-----

MachineClass   Type()  {return CMMIsCartCMM; }

//-----

virtual        CoordSystem(CoordSys csy);
ie             CoordSystem() {return _CoordSys; }

//-----

ie             PartCsyTransformation(const T33EA &tra);
T33EA          PartCsyTransformation();

//-----


protected:

virtual r8      X();           // return machine position in
the
virtual r8      Y();           // selected coordinate
system
virtual r8      Z();
virtual V3      IJK();
```

```

virtual ie X(cr8 x); // move machine to target position
virtual ie Y(cr8 y);
virtual ie Z(cr8 z);
virtual ie IJK(const V3 &ijk);

//-----
};

#endif

9.4.2 \cartcmm\eulerw.cpp
// EulerA.cpp: implementation of the EulerA class.

#include "R33.h"
#include "EulerW.h"

//-----

cr8 R_Delta = 1e-12;

r8 abs (cr8 x);
r8 sind (cr8 x);
r8 cosd (cr8 x);
r8 Acosd (cr8 x);
r8 Atan2d(cr8 y, cr8 x);

//-----

EulerA::EulerA(cR33 &b){ // create Euler from
    _Psi = 0, // rotation matrix
    _Phi = 0;
    s3 = 0,
    c3 = 0,
    c1 = b.Val(3,3);
    _Theta = Acosd(c1);
    s1 = sind(_Theta);
    if (abs(s1) > R_Delta) { // check if Thet() is 0
        s2 = b.Val(1,3)/s1, // no calculate
        c2 = -b.Val(2,3)/s1;
        _Psi = Atan2d(s2, c2);
    }
    ew(_Theta, _Psi, _Phi); // use Thet(), Psi(), 0 to create matrix
    r(ew); -r; // and
}

calcalate Psi() from orig mat // and
R33 rr(b); rr*=r; // and
matrix build from Thet() Psi()
r33 c3 = r.Val(1,1);
r33 s3 = r.Val(2,1);}
else {
    // Thet()==0, Psi()==0 // calculate phi
    c3 = b.Val(1,1);
    s3 = b.Val(2,1);}
    _Phi = Atan2d(s3, c3);}

//-----

void R33::Create(cEulerA &b) {
    r8 psi==0
    c1 = cosd(b.Thet()), // theta==0 &&
    s1 = sind(b.Thet()), // c3
    c2 = cosd(b.Psi()), // -s3
    0
    s2 = sind(b.Psi()), // c3
    0
    c3 = cosd(b.Phi()), // 0
    1
    s3 = sind(b.Phi()); // Mat(1,1) = c2*c3-c1*s2*s3;
    Mat(1,2) = s2*c3+c1*c2*s3;
    Mat(1,3) = s1*s3;
    Mat(2,1) = -c2*s3-c1*s2*c3;
    Mat(2,2) = -s2*s3+c1*c2*c3;
}

```

```

        Mat(2,3) = s1*c3;           //      c2*c3-c1*s2*s3      s2*c3+c1*c2*s3      s1*s3
        Mat(3,1) = s1*s2;           //      -c2*s3-c1*s2*c3    -s2*s3+c1*c2*c3      s1*c3
        Mat(3,2) = -s1*c2;
        Mat(3,3) = c1;            //      s1*s2                  -s1*c2
        c1

//-----

9.4.3 \cartcmmwithrottbl\cartcmmwithrottbl.h

#ifndef AFX_CartCmmWithRotTbl_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00_INCLUDED_
#define AFX_CartCmmWithRotTbl_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00_INCLUDED_

#include "../cartcmm/cartcmm.h"

//-----

class CartCmmWithRotTbl: public CartCMM {
//-----

    Axis                      _RAxis;

//-----

public:                         CartCmmWithRotTbl           ();
virtual ~CartCmmWithRotTbl       () ;

//-----

    Axis*                      RAx() {return &_RAxis; }

//-----

    ie                          SetRZero();
    ie                          AlignPart(const V3 &ijk);

//-----

    virtual char*               Type() {return "CartCMMWithRotTbl";}

//-----

    virtual r8                  X();                                // return machine position in
    virtual r8                  Y();                                // selected coordinate
    system
    virtual r8                  Z();
    virtual r8                  R();
    virtual V3                 IJK();                             // move machine to target position

    virtual ie                  X(cr8 x);
    virtual ie                  Y(cr8 y);
    virtual ie                  Z(cr8 z);
    virtual ie                  R(cr8 r);
    virtual ie                  IJK(const V3 &ijk);

//-----
};

#endif
#endif

```

9.5 \toolchanger

9.5.1 \toolchanger\toolchanger.h

// ToolChanger.h: interface for the ToolChanger class.

```

#ifndef AFX_ToolChanger_H__2418E2DB_F44D_4F25_B290_3EDC4854E112_INCLUDED_
#define AFX_ToolChanger_H__2418E2DB_F44D_4F25_B290_3EDC4854E112_INCLUDED_

#include "ToolAB.h"
//#include <Ary.h>

//-----

class ToolChanger {                                     friend class DME;

```

```

Tool*          _ActTool;
Tool*          _FoundTool;
Tool*          _DefaultTool;
Tool*          _UndefTool;

//-----

Ary<Tool*>      _Tools;
V3              _TransferPosition;

//-----



ToolChanger() {
    /*

String           name     = "DefaultTool"
                _DefaultTool = new Tool(name);

name     = "UndefTool"
_UndefTool = new Tool(name);

name     = "NoTool"
Tool*            tool     = new Tool(name);
                _Tool.Add(tool);

name     = "ReferenceTool"
Tool*            tool     = new Tool(name);
                _Tool.Add(tool);

    */
}

//-----


virtual ~ToolChanger();

//-----


Tool*          ActTool() const {return _ActTool;}
Tool*          FoundTool() const {return _FoundTool;}

//-----


GoToPars*        GoToPar() const {GoToPars* r=ActTool()->GoToPar(); if (r==Nil) r=_DefaultTool->GoToPar(); return r;}
GoToPars*        ABCGoToPar() const {GoToPars* r=ActTool()->ABCGoToPar(); if (r==Nil) r=_DefaultTool->ABCGoToPar(); return r; }

//-----


PtMeasPars*      PtMeasPar() const {PtMeasPars* r=ActTool()->PtMeasPar(); if (r==Nil) r=_DefaultTool->PtMeasPar(); return r;}
PtMeasPars*      ABCPtMeasPar() const {PtMeasPars* r=ActTool()->ABCPtMeasPar(); if (r==Nil) r=_DefaultTool->ABCPtMeasPar(); return r; }

//-----


i4              Howmany(cTag tag) {return _Tools.Len();}

//-----


ie              Qualify(cTag tag) {return _ActTool->Qualify(tag);}
ie              ChangeTool(cTag tag, cString &name);
ie              FindTool(cTag tag, cString &name) {_FoundTool = Find(tag, name); return (_FoundTool==Nil) ? ErrorToolNotFound : ErrorSuccess;}
ie              FindTool(cTag tag, cv3 &ijk) {_FoundTool = Find(tag, ijk); return (_FoundTool==Nil) ? ErrorToolNotFound : ErrorSuccess;}
String          ActToolName(cTag tag) {return _ActTool->Name();}

//-----


void            EnumTools(cTag tag) {
String          name;
    for (i4 i=0; i < _Tools.Len(); i++) {
        name = _Tools[i]->Name();
        /*send name to client*/}
}

```

```

//-----
private:
Tool*          Find(cTag tag, cString &name) /* for(i..) toolname = _Tools[i]->Name()*/;
Tool*          Find(cTag tag, cV3  &ijk)    /* for(i..) toolname = _Tools[i]->Name()*/;

//-----
};

#endif

```

9.5.2 \toolchanger\tool.h

```

// Tool.h: interface for the Tool class.

#ifndef AFX_Tool_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define AFX_Tool_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "String.h"
#include "GoToParams.h"
#include "PtMeasPars.h"
#include "V3.h"
#include "Axis.h"

//-----

class Tool {
friend class ToolChanger;

String          _Name;
i4              _Type;

//-----

GoToPars*        _GoToPars;
GoToPars*        _ABCGoToPars;

//-----

PtMeasPars*      _PtMeasPars;
PtMeasPars*      _ABC_PtMeasPars;

//-----

String          _QualificationArtifact;
i4              _QualificationState;
DateTime        _LastQualificationDate;
ui              _MethodsSupported;

//-----

public:
Tool(cString &name);
virtual ~Tool();

//-----

String          Name() {return _Name;}

//-----

GoToPars*        GoToPar () const {return _GoToPars;}
GoToPars*        ABCGoToPar() const {return _ABCGoToPars;}

//-----

PtMeasPars*      PtMeasPar() const {return _PtMeasPars;}
PtMeasPars*      ABCPtMeasPar() const {return _ABC_PtMeasPars;}

//-----

bool            CanDoGoTo ();
bool            CanDoPtMeas();

//-----

ie              Qualify(cTag tag);

//-----

virtual ie      Align (cTag tag, cV3 &ijk) {return ErrorBadContext;}
virtual void    EnumProp(cTag tag);

```

```

//-----
protected:
virtual r8          A()           {return 0;}
virtual r8          B()           {return 0;}
virtual r8          C()           {return 0;}

virtual ie          A(cr8 a)    {return ErrorSuccess;}
virtual ie          B(cr8 b)    {return ErrorSuccess;}
virtual ie          C(cr8 c)    {return ErrorSuccess;}

//-----
};

#endif

```

9.5.3 \toolchanger\toolab.h

// ToolAB.h: interface for the ToolAB class.

```
#if !defined(AFX_ToolAB_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_)
# define AFX_ToolAB_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_
```

```
#include "Tool.h"
```

```
//-----
```

```
class ToolAB : public Tool {
```

```
Axis          _AAxis;
Axis          _BAxis;
```

```
//-----
```

```
public:        ToolAB(cString &name);
virtual       ~ToolAB();
```

```
//-----
```

```
ie           Align (cTag tag, cV3 &ijk);
void         EnumProp(cTag tag);
```

```
//-----
```

```
r8           A();
r8           B();
```

```
ie           A(cr8 a);
ie           B(cr8 b);
```

```
//-----
```

```
//-----  
};  
#endif
```

9.5.4 \toolchanger\toolabc.h

// ToolABC.h: interface for the ToolABC class.

```
#if !defined(AFX_ToolABC_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_)
# define AFX_ToolABC_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_
```

```
#include "ToolAB.h"
```

```
//-----
```

```
class ToolABC : public ToolAB {
```

```
Axis          _CAxis;
```

```
//-----
```

```
public:        ToolABC(const String &name);
virtual       ~ToolABC();
```

```
//-----
```

```

ie Align (cTag tag, cV3 &ijk);
void EnumProp(cTag tag);

//-----
r8 C();
ie C(cr8 c);

//-----
};

#endif

9.5.5 \toolchanger\gotoparams.h
// GoToPars.h: interface for the GoToPars class.

#ifndef _AFX_GOTOPIRS_H_
#define _AFX_GOTOPIRS_H_ _INCLUDED_

#include "../lib/String.h"
#include "Param.h"

//-----

class GoToPars {

Param _Speed;
Param _Accel;

//-----

public: GoToPars();
virtual ~GoToPars();

//-----

r8 MinSpeed() const {return _Speed.Min();}
r8 Speed () const {return _Speed.Val();}
r8 MaxSpeed() const {return _Speed.Max();}

bool CanChangeSpeed() const {return _Speed.CanChange();}
ie Speed(cr8 s) {return _Speed.Val(s);}

//-----

r8 MinAccel() const {return _Accel.Min();}
r8 Accel () const {return _Accel.Val();}
r8 MaxAccel() const {return _Accel.Max();}

bool CanChangeAccel() const {return _Accel.CanChange();}
ie Accel(cr8 s) {return _Accel.Val(s);}

//-----

void EnumProp();

//-----
};

#endif

```

9.5.6 \toolchanger\ptmeaspars.h

// PtMeasPars.h: interface for the PtMeasPars class.

```

#ifndef _AFX_PTMEASPIRS_H_
#define _AFX_PTMEASPIRS_H_ _INCLUDED_

#include "../lib/String.h"
#include "../lib/DateTime.h"
#include "GoToParams.h"

//-----

class PtMeasPars {

Param _Approach;
r8 _Search;
r8 _Retract;
GoToPars _Move;

```

```

//-----
public:      PtMeasPars();
virtual ~PtMeasPars();

//-----

r8          MinSpeed()           {return _Move.MinSpeed();}
r8          Speed ()            {return _Move.Speed();}
r8          MaxSpeed()          {return _Move.MaxSpeed();}
ie          Speed (cr8 s)       {return _Move.Speed(s);}

//-----

r8          MinAccel()          {return _Move.MinAccel();}
r8          Accel ()             {return _Move.Accel();}
r8          MaxAccel()          {return _Move.MaxAccel();}
ie          Accel (cr8 s)       {return _Move.Accel(s);}

//-----

void        EnumProp();

//-----
};

#endif

```

9.5.7 \toolchanger\param.h

// Param.h: interface for the Param class.

```

#ifndef !defined(AFX_Param_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_)
#define AFX_Param_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "IppTypeDef.h"
#include <IppErrorCodes.h>

r8          min(cr8 a, cr8 b);
r8          max(cr8 a, cr8 b);

//-----

class Param {

r8          _Min;
r8          _Val;
r8          _Max;
bool        _CanChange;

//-----

public:      Param()  {_Min=-10000; _Val=0; _Max=10000; _CanChange=Tr;}
virtual ~Param();

//-----

r8          Min ()           const {return _Min;}
r8          Val ()            const {return _Val;}
r8          Max ()            const {return _Max;}
bool        CanChange()     const {return _CanChange || (_Max-_Min)<=0;}
void        EnumProp ();

//-----

ie          Val(cr8 v) {
    errcod = ErrorSuccess;
    r    = CanChange();
    if (r) {
        r = v >= _Min && v <= _Max;
    if (r) {
        _Val=v;
    else {
        _Val = min(v, _Max);
        _Val = max(_Val, _Min);
    errcod = (v < _Min) ? ErrorParamTooSmall : ErrorParamTooLarge;}}
    else {
        errcod = ErrorParamCannotBeChanged;}}
```

```

        return errcod; }

//-----
private:
void          Min           (cr8 v) { _Min=v; }
void          Max           (cr8 v) { _Max=v; }
void CanChange(cbo v) { _CanChange=v; }

//-----
};

#endif

```

9.6 Most important of lib

9.6.1 \lib\axis.h

```

// Axis.h: interface for the Axis class.

#ifndef AFX_AXIS_H_B3DA30C7_5415_11D3_A481_0000F87ABD00_INCLUDED_
#define AFX_AXIS_H_B3DA30C7_5415_11D3_A481_0000F87ABD00_INCLUDED_

#include "IppTypeDef.h"

//-----

class Axis {

//-----

    enum     AxisType {Lin=1, Rot=2};

    char          _Name[8];
    AxisType _Type;
    r8            _MinPos;
    r8            _ActPos;
    r8            _MaxPos;
    r8            _Pitch;
    r8            _Temperature;
    bool          _IsControlled;
    bool          _IsHomed;

//-----

public:      Axis();
virtual ~Axis(){};

//-----

    i4            Type()           const {return _Type;}
    r8            MinPos()         const {return _MinPos;}
    r8            MaxPos()         const {return _MaxPos;}
    r8            Pitch()          const {return _MaxPos;}
    r8            Temperature()    const {return _Temperature;}

//-----

    static void    EnumProp();           // Name,          c*8
                                            // Type,          i4
                                            // MinPos,        r8
                                            // MaxPos,        r8
                                            // Temperature,   r8

//-----
};

#endif

```

9.6.2 \lib\eulerw.h

```

// EulerA.h: interface for the EulerA class.

#ifndef AFX_EULER_A_H_0E096DA3_5537_11D3_84A8_0000F87ADB6B_INCLUDED_
#define AFX_EULER_A_H_0E096DA3_5537_11D3_84A8_0000F87ADB6B_INCLUDED_

#include "IppTop.h"

//-----

```

```

class EulerA {
    r8          _Theta;           // Euler angel in degree
    r8          _Psi;
    r8          _Phi;

//-----
public:             EulerA();
                    EulerA(cr8 theta, cr8 psi, cr8 phi);
                    EulerA(cR33 &b);
virtual            ~EulerA();
//-----

r8          Tht() const {return _Theta;}
r8          Psi() const {return _Psi;}
r8          Phi() const {return _Phi;}

//-----
};

//-----
#endif

```

9.6.3 \lib\tag.h

// KTag.h: interface for the KTag class.

```

#ifndef AFX_KTag_H__001F2611_6298_11D3_A49B_0000F87ABD00__INCLUDED_
#define AFX_KTag_H__001F2611_6298_11D3_A49B_0000F87ABD00__INCLUDED_

#include <IppTypeDef.h>

//-----

class Tag {
    static
    i4          _TagCounter;           // static tag
    counter
    i4          _Tag;                 // tag

//-----

public:             Tag(ci4 i)           {_Tag = i;}
virtual            ~Tag()           {}}

//-----

i4          Val();                {return _Tag;}           //
i4          NewTag();           //

create new tag *** for client use only

//-----
};

#endif

```

9.6.4 \lib\ippypedef.h

// This is the global type definition file

```

#ifndef _IppTypeDefDefined
#define _IppTypeDefDefined

//-----

typedef      unsigned   char   uc;           // define some shortcuts
typedef      const     unsigned   char   cuc;           // for type definitions

typedef      char       char   ch;
typedef      const     char       char   cc;

typedef      unsigned   short  ui2;
typedef      signed    short  i2;
typedef      const     signed    short  ci2;
typedef      const     unsigned   short  cui2;

typedef      signed    int    i4;
typedef      const     signed    int    ci4;

```

```

typedef signed int ie; // error codes
typedef const signed int cie;

typedef unsigned int cui;
typedef const unsigned int ui;

typedef unsigned int ich;
typedef const unsigned int cich;

typedef double r8;
typedef const double cr8;
typedef float r4;
typedef const float cr4;

typedef bool bo;
typedef const bool cbo;

//-----
#define Fa false // define boolean shortcuts
#define Tr true
#define Nil 0

//-----
#endif

```

9.6.5 \lib\ippbaseclasses.h

```

#define _IppBaseClassesDefined

//-----

class String; typedef const String cString;
// data base object

class Tag; typedef const Tag cTag;
// data base object

class ETag; typedef const ETag cETag;
// data base object

class GoToPars; typedef const GoToPars cGoToPars;
// data base object

class PtMeasPars; typedef const PtMeasPars cPtMeasPars;

//-----

class V3; typedef const V3 cV3;
// data base object

class M33; typedef const M33 cM33;
// data base object

class R33; typedef const R33 cR33;
// data base object

class T33; typedef const T33 cT33;
// data base object

class T33EA; typedef const T33EA cT33EA;
// data base object

class EulerA; typedef const EulerA cEulerA;
// data base object

//-----

class Axis; typedef const Axis cAxis;
class Part; typedef const Part cPart;

//-----

enum ErrorSeverity; typedef const ErrorSeverity cErrorSeverity;
enum ErrorCode; typedef const ErrorCode cErrorCode;

//-----
#endif

```